ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES
RAJAMPET
(Autonomous)

Department of Artificial Intelligence & Machine Learning

Lecture Notes

Name of the Faculty: **Dr.T.Harikrishna** **Class:** **IVYear I Semester**

Branch and Section: **AIML** **Course code:** **20A57GT**

Name of the Course: Cloud Computing

Unit-I

Unit-1

When computing resources in distant data centers are used rather than local computing systems, we talk about network-centric computing and network-centric content. Advancements in networking and other areas are responsible for the acceptance of the two new computing models and led to the grid computing movement in the early 1990s and, since 2005, to utility computing and cloud computing.

In utility computing the hardware and software resources are concentrated in large data centers and users can pay as they consume computing, storage, and communication resources. Utility computing often requires a cloud-like infrastructure, but its focus is on the business model for providing the computing services. Cloud computing is a path to utility computing embraced by major IT companies such as Amazon, Apple, Google, HP, IBM, Microsoft, Oracle, and others.

Cloud computing delivery models, deployment models, defining attributes, resources, and organization of the infrastructure discussed in this chapter are summarized in Figure 1.1. There are three cloud delivery models: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-aService (IaaS), deployed as public, private, community, and hybrid clouds.

The defining attributes of the new philosophy for delivering computing services are as follows:

Cloud computing uses Internet technologies to offer elastic services. The term elastic computing refers to the ability to dynamically acquire computing resources and support a variable workload.

A cloud service provider maintains a massive infrastructure to support elastic services.

• The resources used for these services can be metered and the users can be charged only for the resources they use.

• Maintenance and security are ensured by service providers.

• Economy of scale allows service providers to operate more efficiently due to specialization and centralization.

• Cloud computing is cost-effective due to resource multiplexing; lower costs for the service provider are passed on to the cloud users.

• The application data is stored closer to the site where it is used in a device- and location-independent.

Cloud computing is a technical and social reality and an emerging technology. At this time, one

can only speculate how the infrastructure for this new paradigm will evolve and what applications will migrate to it. The economical, social, ethical, and legal implications of this shift in technology, in which users rely on services provided by large data centers and store private data and software on systems they do not control, are likely to be significant. Manner potentially, this data storage strategy increases reliability and security and, at the same time, it lowers communication cost.

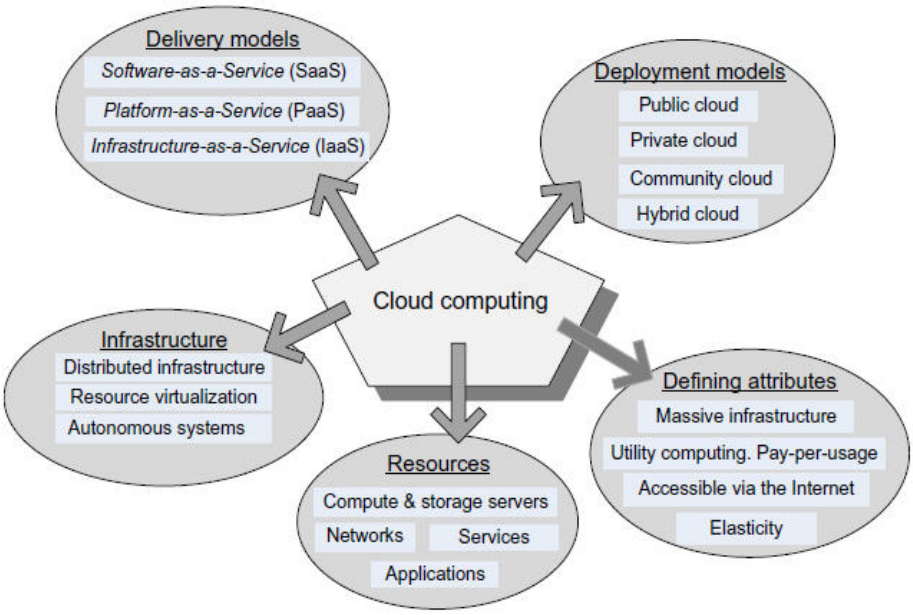


FIGURE 1.1

Cloud computing: Delivery models, deployment models, defining attributes, resources, and organization of the infrastructure.

Scientific and engineering applications, data mining, computational financing, gaming, and social networking as well as many other computational and data-intensive activities can benefit from cloud computing.

In early 2011 Apple announced the iCloud, a network-centric alternative for storing content such as music, videos, movies, and personal information; this content was previously confined to personal devices such as workstations, laptops, tablets, or smartphones. The obvious advantage of network centric content is the accessibility of information from any site where users can connect to the Internet.

Network-centric computing and network-centric content

The concepts and technologies for network-centric computing and content evolved through the years and led to several large-scale distributed system developments:

The Web and the semantic Web are expected to support composition of services

The Grid, initiated in the early 1990s by National Laboratories and Universities, is used primarily for applications in the area of science and engineering.

Computer clouds, promoted since 2005 as a form of service-oriented computing by large IT companies, are used for enterprise computing, high-performance computing, Web hosting, and storage for network-centric content.

The semantic Web2 is an effort to enable laypeople to more easily find, share, and combine information available on the Web. In this vision, the information can be readily interpreted by machines, so machines can perform more of the tedious work involved in finding, combining, and acting upon information on the Web. Several technologies are necessary to provide a formal description of concepts, terms, and relationships within a given knowledge domain; they include the Resource Description Framework (RDF), a variety of data interchange formats, and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL).

A computing grid is a distributed system consisting of a large number of loosely coupled, heterogeneous, and geographically dispersed systems in different administrative domains. The term computing grid is a metaphor for accessing computer power with similar ease as we access power provided by the electric grid.

Cloud computing is a technology largely viewed as the next big step in the development and deployment of an increasing number of distributed applications. The companies promoting cloud computing seem to have learned the most important lessons from the grid movement. Computer clouds are typically homogeneous. An entire cloud shares the same security, resource management, cost and other policies,and last but not least, it targets enterprise computing. These are some of the reasons that several agencies of the US Government, including Health and Human Services (HHS), the Centers for Disease Control (CDC), the National Aeronautics and Space Administration (NASA), the Navy's Next Generation Enterprise Network (NGEN), and the Defense Information Systems Agency (DISA), have launched cloud computing initiatives and conduct actual system development intended to improve the efficiency and effectiveness of their information processing needs.

The term content refers to any type or volume of media, be it static or dynamic, monolithic or modular, live or stored, produced by aggregation, or mixed. Information is the result of functions

applied to content. The creation and consumption of audio and visual content are likely to transform the Internet to support increased quality in terms of resolution, frame rate, color depth, and stereoscopic information, and it seems reasonable to assume that the Future Internet3 will be content-centric. Content-centric routing will allow users to fetch the desired data from the most suitable location in terms of network latency or download time. There are also some challenges, such as providing secure services for content manipulation, ensuring global rights management, control over unsuitable content, and reputation management.

Network-centric computing and network-centric content share a number of characteristics:

Most applications are data-intensive. Computer simulation becomes a powerful tool for scientific research in virtually all areas of science, from physics, biology, and chemistry to archeology. Sophisticated tools for computer-aided design, such as Catia (Computer Aided Three-dimensional Interactive Application), are widely used in the aerospace and automotive industries.

Virtually all applications are network-intensive. Indeed, transferring large volumes of data requires high-bandwidth networks; parallel computing, computation steering,4 and data streaming are examples of applications that can only run efficiently on low-latency networks.

The systems are accessed using thin clients running on systems with limited resources.

The infrastructure supports some form of workflow management. Indeed, complex computational tasks require coordination of several applications; composition of services is a basic tenet of Web 2.0.

The advantages of network-centric computing and network-centric content paradigms are, at the same time, sources for concern;

Computing and communication resources (CPU cycles, storage, network bandwidth) are shared and resources can be aggregated to support data-intensive applications. Multiplexing leads to a higher resource utilization; indeed, when multiple applications share a system, their peak demands for resources are not synchronized and the average system utilization increases.

Data sharing facilitates collaborative activities. Indeed, many applications in science, engineering, and industrial, financial, and governmental applications require multiple types of analysis of shared data sets and multiple decisions carried out by groups scattered around the globe. Open software development sites are another example of such collaborative activities.

Cost reduction. Concentration of resources creates the opportunity to pay as you go for computing and thus eliminates the initial investment and reduces significantly the maintenance and operation costs of the local computing infrastructure.

User convenience and elasticity that is the ability to accommodate workloads with very large peakto-average ratios.

Peer-to-peer systems

System administrators enforce security rules and control the allocation of physical rather than virtual resources. In all models of network-centric computing prior to utility computing, a user maintains direct control of the software and the data residing on remote systems.

his user-centric model, in place since the early 1960s, was challenged in the 1990s by the peer-topeer (P2P) model. P2P systems can be regarded as one of the precursors of today's clouds. This new model for distributed computing promoted the idea of low-cost access to storage and central processing unit (CPU) cycles provided by participant systems; in this case, the resources are located in different administrative domains. Often the P2P systems are self-organizing and decentralized, whereas the servers in a cloud are in a single administrative domain and have a central management.

P2P systems exploit the network infrastructure to provide access to distributed computing resources. Decentralized applications developed in the 1980s, such as Simple Mail Transfer Protocol (SMTP), a protocol for email distribution, and Network News Transfer Protocol (NNTP), an application protocol for dissemination of news articles, are early examples of P2P systems.

The P2P model represents a significant departure from the client-server model, the cornerstone of distributed applications for several decades. P2P systems have several desirable properties.

They require a minimally dedicated infrastructure, since resources are contributed by the participating systems.

• They are highly decentralized.

• They are scalable; the individual nodes are not required to be aware of the global state.

• They are resilient to faults and attacks, since few of their elements are critical for the delivery of service and the abundance of resources can support a high degree of replication.

• Individual nodes do not require excessive network bandwidth the way servers used in case of the client-server model do.

• Last but not least, the systems are shielded from censorship due to the dynamic and often unstructured system architecture.

Many groups from industry and academia rushed to develop and test new ideas, taking advantage of the fact that P2P applications do not require a dedicated infrastructure. Applications such as Chord [334] and Credence [366] address issues critical to the effective operation of decentralized systems. **Chord** is a distributed lookup protocol to identify the node where a particular data item is stored. The routing tables are distributed and, whereas other algorithms for locating an object require the nodes to be aware of most of the nodes of the network, Chord maps a key related to an object to a node of the network using routing information about a few nodes only.

Credence is an object reputation and ranking scheme for large-scale P2P file-sharing systems. Reputation is of paramount importance for systems that often include many unreliable and malicious nodes.In the decentralized algorithm used by Credence, each client uses local information to evaluate the reputation of other nodes and shares its own assessment with its neighbors. The credibility of a node depends only on the votes it casts; each node computes the reputation of another node based solely on the degree of matching with its own votes and relies on like-minded peers. **Overcite** is a P2P application to aggregate documents based on a three-tier design.

Skype has a central site to maintain user accounts; users sign in and pay for specific activities at this site. The controller for a BOINC platform maintains membership and is involved in task distribution to participating systems. The nodes with abundant resources in systems without any centralized infrastructure often act as super nodes and maintain information useful to increasing the system efficiency, such as indexes of the available content.

Regardless of the architecture, P2P systems are built around an overlay network, a virtual network superimposed over the real network. Methods to construct such an overlay, consider a graph G = (V, E), where V is the set of N vertices and E is the set of links between them.

Structured overlay networks use key-based routing (KBR); given a starting node v0 and a key k, the function KBR(v0, k) returns the path in the graph from v0 to the vertex with key k.Epidemic algorithms discussed in Section 7.12 are often used by unstructured overlays to disseminate network topology.

Cloud computing: an old idea whose time has come

Once the technological elements were in place, it was only a matter of time until the

economical advantages of cloud computing became apparent. Due to the economy of scale, large data centers centers with more than 50,000 systems – are more economical to operate than medium-sized centers that have around 1,000 systems. Large data centers equipped with commodity computers experience a five to seven times decrease of resource consumption, including energy, compared to medium-sized centers.

The term computer cloud is overloaded, since it covers infrastructures of different sizes, with different management and different user populations. Several types of cloud are envisioned.

Private cloud. The infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on or off the premises of the organization.

• **Community cloud**. The infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premises or off premises.

• **Public cloud**. The infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

• **Hybrid cloud**. The infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

A private cloud could provide the computing resources needed for a large organization, such as a research institution, a university, or a corporation. The argument that a private cloud does not support utility computing is based on the observation that an organization has to invest in the infrastructure and a user of a private cloud pays as it consumes resources.

A nonexhaustive list of reasons for the success of cloud computing includes these points:

Cloud computing is in a better position to exploit recent advances in software, networking, storage,and processor technologies. Cloud computing is promoted by large IT companies where these new technological developments take place, and these companies have a vested interest in promoting the new technologies.

• A cloud consists of a homogeneous set of hardware and software resources in a single administrative domain. In this setup, security, resource management, fault tolerance, and quality

of service are less challenging than in a heterogeneous environment with resources in multiple administrative domains.

Cloud computing is focused on enterprise computing; its adoption by industrial organizations, financial institutions, healthcare organizations, and so on has a potentially huge impact on the economy.

• A cloud provides the illusion of infinite computing resources; its elasticity frees application designers from the confinement of a single system.

• A cloud eliminates the need for up-front financial commitment, and it is based on a pay-as-you-go approach. This has the potential to attract new applications and new users for existing applications,

In spite of the technological breakthroughs that have made cloud computing feasible, there are still major obstacles for this new technology; these obstacles provide opportunity for research. We list a few of the most obvious obstacles.

Availability of service A partial answer to this question is provided by service-level agreements (SLAs).6 A temporary fix with negative economical implications is over provisioning, that is, having enough resources to satisfy the largest projected demand.

Vendor lock-in. Once a customer is hooked to one provider, it is hard to move to another. The standardization efforts at National Institute of Standards and Technology (NIST) attempt to address this problem.

Data confidentiality and auditability. This is indeed a serious problem.

Data transfer bottlenecks. Many applications are data-intensive. A very important strategy is to store the data as close as possible to the site where it is needed.

Performance unpredictability. This is one of the consequences of resource sharing. Strategies for performance isolation.

Elasticity, the ability to scale up and down quickly. New algorithms for controlling resource allocation and workload placement are necessary. Autonomic computing based on self-organization and self-management seems to be a promising avenue.

Cloud computing delivery models and services

According to the NIST reference model in Figure 1.2 the entities involved in cloud computing are the **service consumer**, the entity that maintains a business relationship with and uses service from service providers; **the service provider**, the entity responsible for making a service

available to service consumers; **the carrier**, the intermediary that provides connectivity and transport of cloud services between providers and consumers; **the broker**, an entity that manages the use, performance, and delivery of cloud services and negotiates relationships between providers and consumers; and the **auditor**, a party that can conduct independent assessment of cloud services, information system operations, performance, and security of the cloud implementation. **An audit** is a systematic evaluation of a cloud system that measures how well it conforms to a set of established criteria.

cloud security, a privacy-impact audit evaluates cloud privacy assurance, and a performance audit evaluates cloud performance.

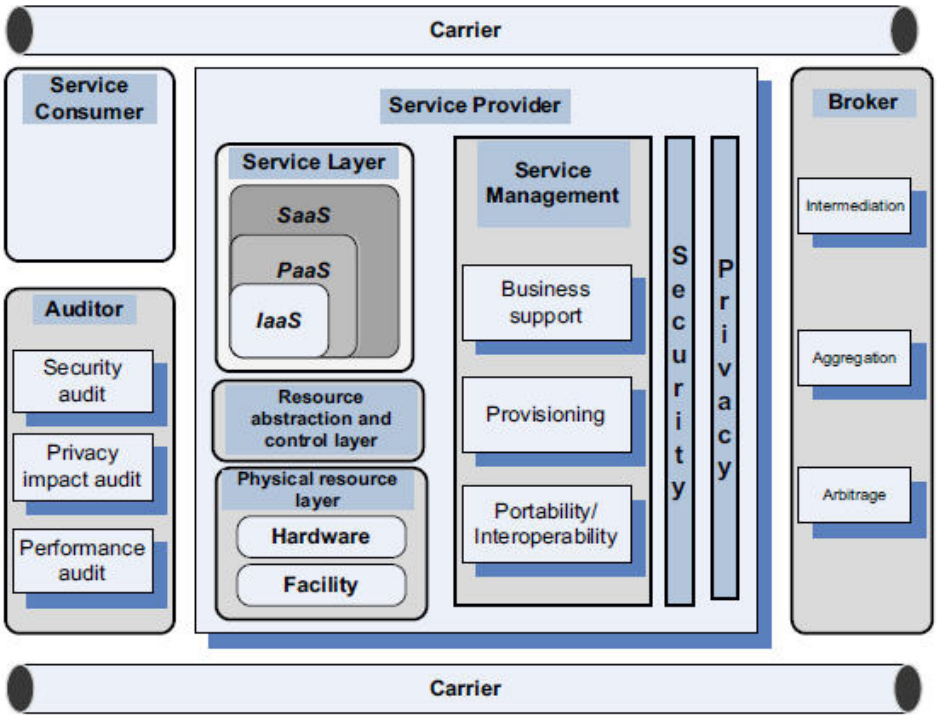


FIGURE 1.2

The entities involved in service-oriented computing and, in particular, in cloud computing, according to NIST. The carrier provides connectivity among service providers, service consumers, brokers, and auditors.

Figure 1.3 presents the structure of the three delivery models, SaaS, PaaS, and IaaS, according to the Cloud Security Alliance.

Software-as-a-Service (SaaS) gives the capability to use applications supplied by the service provider in a cloud infrastructure. The applications are accessible from various client devices through a thin-client. Interface such as a Web browser (e.g., Web-based email). The user does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings. Services offered include:

• Enterprise services such as workflow management, groupware and collaborative, supply chain, communications, digital signature, customer relationship management (CRM), desktop software, financial management, geo-spatial, and search.

• Web 2.0 applications such as metadata management, social networking, blogs, wiki services, and portal services.

The SaaS is not suitable for applications that require real-time response or those for which data is not allowed to be hosted externally. The most likely candidates for SaaS are applications for which:

• Many competitors use the same product, such as email.

• Periodically there is a significant peak in demand, such as billing and payroll.

• There is a need for Web or mobile access, such as mobile sales management software.

• There is only a short-term need, such as collaborative software for a project.

Platform-as-a-Service (PaaS) gives the capability to deploy consumer-created or acquired applications using programming languages and tools supported by the provider. The user does not manage or control the underlying cloud infrastructure, including network, servers, operating systems, or storage. The user has control over the deployed applications and, possibly, over the application hosting environment configurations.

PaaS is not particulary useful when the application must be portable, when proprietary programming languages are used, or when the underlaying hardware and software must be customized to improve the performance of the application. The major PaaS application areas are in software development where multiple developers and users collaborate and the deployment and testing services should be automated.

Infrastructure-as-a-Service (IaaS) is the capability to provision processing, storage, networks, and other fundamental computing resources; the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage

or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of some networking components, such as host firewalls. Services offered by this delivery model include: server hosting, Web servers, storage, computing hardware, operating systems, virtual instances, load balancing, Internet access, and bandwidth provisioning.

The IaaS cloud computing delivery model has a number of characteristics, such as the fact that the resources are distributed and support dynamic scaling, it is based on a utility pricing model and variable cost, and the hardware is shared among multiple users. This cloud computing model is particularly useful when the demand is volatile and a new business needs computing resources and does not want to invest in a computing infrastructure or when an organization is expanding rapidly.

A number of activities are necessary to support the three delivery models; they include:

1. Service management and provisioning, including virtualization, service provisioning, call center operations management, systems management, QoS management, billing and accounting, asset management, SLA management, technical support, and backups.
2. Security management, including ID and authentication, certification and accreditation, intrusion prevention, intrusion detection, virus protection, cryptography, physical security, incident response, access control, audit and trails, and firewalls.
3. Customer services such as customer assistance and online help, subscriptions, business intelligence, reporting, customer preferences, and personalization.
4. Integration services, including data management and development.

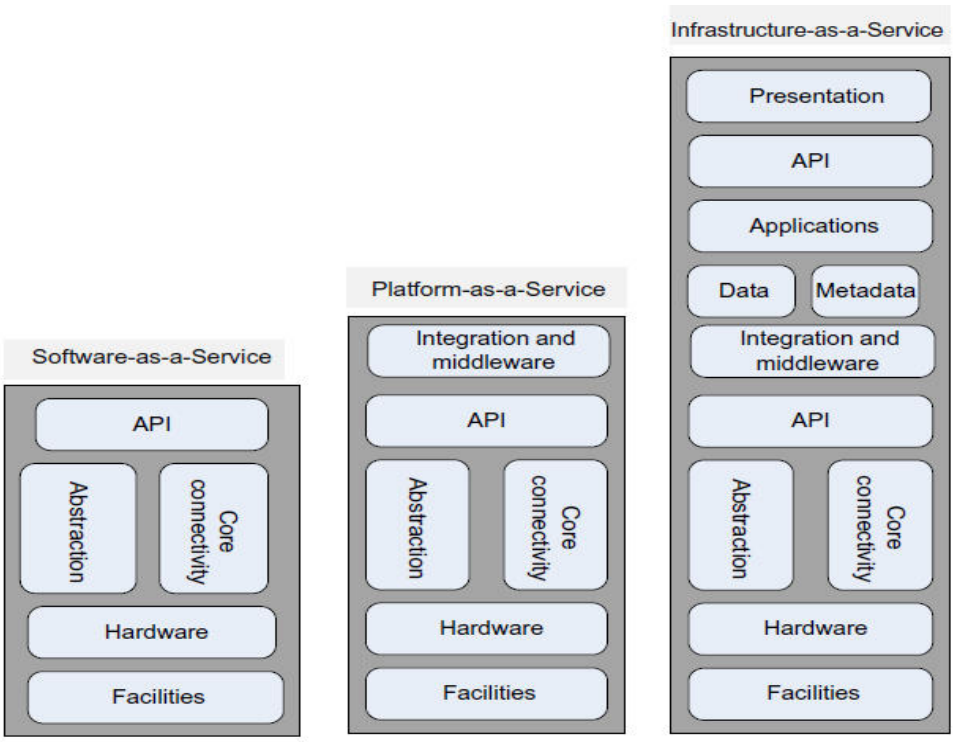


FIGURE 1.3

Ethical issues in cloud computing

Cloud computing is based on a paradigm shift with profound implications for computing ethics. The main elements of this shift are: (i) the control is relinquished to third-party services; (ii) the data is stored on multiple sites administered by several organizations; and (iii) multiple services interoperate across the network.

Unauthorized access, data corruption, infrastructure failure, and service unavailability are some of the risks related to relinquishing the control to third-party services; moreover, whenever a problem occurs, it is difficult to identify the source and the entity causing it. Systems can span the boundaries of multiple organizations and cross security borders, a process called deperimeterization. As a result of deperimeterization, "not only the border of the organization's IT infrastructure blurs, also the border of the accountability becomes less clear".

The complex structure of cloud services can make it difficult to determine who is responsible in case something undesirable happens.

Ubiquitous and unlimited data sharing and storage among organizations test the self-determination of information, the right or ability of individuals to exercise personal control over

the collection, and use and disclosure of their personal data by others; this tests the confidence and trust in today's evolving information society.

Cloud service providers have already collected petabytes of sensitive personal information stored in data centers around the world. The acceptance of cloud computing therefore will be determined by privacy issues addressed by these companies and the countries where the data centers are located.

Accountability is a necessary ingredient of cloud computing; adequate information about how data is handled within the cloud and about allocation of responsibility are key elements for enforcing ethics rules in cloud computing. Recorded evidence allows us to assign responsibility; but there can be tension between privacy and accountability, and it is important to establish what is being recorded and who has access to the records.

Unwanted dependency on a cloud service provider, the so-called vendor lock-in, is a serious concern, and the current standardization efforts at NIST attempt to address this problem. Another concern for users is a future with only a handful of companies that dominate the market and dictate prices and policies.

Cloud vulnerabilities

Clouds are affected by malicious attacks and failures of the infrastructure (e.g., power failures). Such events can affect Internet domain name servers and prevent access to a cloud or can directly affect the clouds.

The recovery from the failure took a very long time and exposed a range of problems. For example, one of the 10 centers failed to switch to backup generators before exhausting the power that could be supplied by uninterruptible power supply (UPS) units. AWS uses "control planes" to allow users to switch to resources in a different region, and this software component also failed. The booting process was faulty and extended the time to restart EC2 (Elastic Computing) and EBS (Elastic Block Store) services. Another critical problem was a bug in the elastic load balancer (ELB), which is used to route traffic to servers with available capacity. A similar bug affected the recovery process of the Relational Database Service (RDS). This event brought to light "hidden" problems that occur only under special circumstances,

A cloud application provider, a cloud storage provider, and a network provider could implement different policies, and the unpredictable interactions between load-balancing and other reactive mechanisms could lead to dynamic instabilities. The unintended coupling of independent

controllers that manage the load, the power consumption, and the elements of the infrastructure could lead to undesirable feedback and instability similar to the ones experienced by the policy-based routing in the Internet Border Gateway Protocol (BGP).

Clustering the resources in data centers located in different geographical areas is one of the means used today to lower the probability of catastrophic failures. This geographic dispersion of resources could have additional positive side effects; it can reduce communication traffic and energy costs by dispatching the computations to sites where the electric energy is cheaper, and it can improve performance by an intelligent and efficient load-balancing strategy.

Major challenges faced by cloud computing

The most significant challenge is security gaining the trust of a large user base is critical for the future of cloud computing. It is unrealistic to expect that a public cloud will provide a suitable environment for all applications. Highly sensitive applications related to the management of the critical infrastructure, healthcare applications, and others will most likely be hosted by private clouds. Many real-time applications will probably still be confined to private clouds.

The SaaS model faces similar challenges as other online services required to protect private information, such as financial or healthcare services. In this case a user interacts with cloud services through a well-defined interface; thus, in principle it is less challenging for the service provider to close some of the attack channels. Still, such services are vulnerable to DoS attack and the users are fearful of malicious insiders. Data in storage is most vulnerable to attack, so special attention should be devoted to the protection of storage servers. Data replication necessary to ensure continuity of service in case of storage system failure increases vulnerability. Data encryption may protect data in storage, but eventually data must be decrypted for processing, and then it is exposed to attack.

Virtualization is a critical design option for this model, but it exposes the system to new sources of attack. The trusted computing base (TCB) of a virtual environment includes not only the hardware and the hypervisor but also the management operating system.

It seems reasonable to expect that such a complex system can only function based on self-management principles. But self-management and self-organization raise the bar for the implementation of logging and auditing procedures critical to the security and trust in a provider of cloud computing service.

The last major challenge we want to address is related to interoperability and standardization. Vendor lock-in, the fact that a user is tied to a particular cloud service provider, is a major concern for cloud users.

Cloud Infrastructure

Private clouds are an alternative to public clouds. Open-source cloud computing platforms such as Eucalyptus , OpenNebula, Nimbus, and OpenStack can be used as a control infrastructure for a private cloud.

Several other IT companies are also involved in cloud computing. IBM offers a cloud computing platform, IBM Smart Cloud, which includes servers, storage, and virtualization components for building private and hybrid cloud computing environments. In October 2012 it was announced that IBM had teamed

up with AT&T to give customers access to IBM's cloud infrastructure over AT&T's secure private lines.In 2011 HP announced plans to enter the cloud computing club. Oracle announced its entry to enterprise computing in the early 2012. The Oracle Cloud is based on Java, SQL standards, and software systems such as Exadata, Exalogic, WebLogic, and Oracle Database. Oracle plans to offer application and platform services. Some of these services are Fusion HCM (Human Capital Management), Fusion CRM (Customer Relation Management), and Oracle Social Network; the platform services are based on Java and SQL

Cloud computing at Amazon

In mid-2000 Amazon introduced Amazon Web Services (AWS), based on the IaaS delivery model. In this model the cloud service provider offers an infrastructure consisting of compute and storage servers interconnected by high-speed networks that support a set of services to access these resources. An application developer is responsible for installing applications on a platform of his or her choice and managing the resources provided by Amazon.

Amazon Web Services. Amazon was the first provider of cloud computing; it announced a limited public beta release of its Elastic Computing platform called EC2 in August 2006. Figure 3.1 shows the palette of AWS services accessible via the Management Console in late 2011.

Elastic Compute Cloud (EC2)1 is a Web service with a simple interface for launching instances of an application under several operating systems, such as several Linux distributions, Microsoft Windows Server 2003 and 2008, OpenSolaris, FreeBSD, and NetBSD.

An instance is created either from a predefined Amazon Machine Image (AMI) digitally signed

and stored in S3 or from a user-defined image. The image includes the operating system, the run-time environment, the libraries, and the application desired by the user. AMI images create an exact copy of the original image but without configuration-dependent information such as the hostname or the MAC address. A user can: (i) Launch an instance from an existing AMI and terminate an instance; (ii) start and stop an instance; (iii) create a new image; (iv) add tags to identify an image; and (v) reboot an instance.

EC2 allows the import of virtual machine images from the user environment to an instance through a facility called VM import. It also automatically distributes the incoming application traffic among multiple instances using the elastic load-balancing facility. EC2 associates an elastic IP address with an account; this mechanism allows a user to mask the failure of an instance and remap a public IP address to any instance of the account without the need to interact with the software support team.

Simple Storage System (S3) is a storage service designed to store large objects. It supports a minimal set of functions: write, read, and delete.S3 allows an application to handle an unlimited number of objects ranging in size from one byte to five terabytes. An object is stored in a bucket and retrieved via a unique developer-assigned key. A bucket can be stored in a region selected by the user. S3 maintains the name, modification time, an access control list, and up to four kilobytes of user-defined metadata for each object. The object names are global. Authentication mechanisms ensure that data is kept secure; objects can be made public, and rights can be granted to other users.

S3 supports PUT, GET, and DELETE primitives to manipulate objects but does not support primitives to copy, rename, or move an object from one bucket to another. Appending to an object requires a read followed by a write of the entire object.

Elastic Block Store (EBS) provides persistent block-level storage volumes for use with Amazon EC2 instances. A volume appears to an application as a raw, unformatted, and reliable physical disk; the size of the storage volumes ranges from one gigabyte to one terabyte. The volumes are grouped together in availability zones and are automatically replicated in each zone. An EC2 instance may mount multiple volumes, but a volume cannot be shared among multiple instances. The EBS supports the creation of snapshots of the volumes attached to an instance and then uses them to restart an instance. The storage strategy provided by EBS is suitable for database applications, file systems, and applications using raw data devices.

Simple DB is a non-relational data store that allows developers to store and query data items via Web services requests. It supports store-and-query functions traditionally provided only by relational databases. Simple DB creates multiple geographically distributed copies of each data item and supports high-performance Web applications; at the same time, it automatically manages infrastructure provisioning, hardware and software maintenance, replication and indexing of data items, and performance tuning.

Simple Queue Service (SQS) is a hosted message queue. SQS is a system for supporting automated workflows; it allows multiple Amazon EC2 instances to coordinate their activities by sending and receiving SQS messages. Any computer connected to the Internet can add or read messages without any installed software or special firewall configurations.

Cloud Watch is a monitoring infrastructure used by application developers, users, and system administrators to collect and track metrics important for optimizing the performance of applications and for increasing the efficiency of resource utilization. Without installing any software, a user can monitor approximately a dozen preselected metrics and then view graphs and statistics for these metrics.

When launching an Amazon Machine Image (AMI), a user can start the CloudWatch and specify the type of monitoring. Basic Monitoring is free of charge and collects data at five-minute intervals for up to 10 metrics; Detailed Monitoring is subject to a charge and collects data at one-minute intervals. This service can also be used to monitor the latency of access to EBS volumes, the available storage space for RDS DB instances, the number of messages in SQS, and other parameters of interest for applications.

Virtual Private Cloud (VPC) provides a bridge between the existing IT infrastructure of an organization and the AWS cloud. The existing infrastructure is connected via a virtual private network (VPN) to a set of isolated AWS compute resources. VPC allows existing management capabilities such as security services, firewalls, and intrusion detection systems to operate seamlessly within the cloud.

Auto Scaling exploits cloud elasticity and provides automatic scaling of EC2 instances. The service supports grouping of instances, monitoring of the instances in a group, and defining triggers and pairs of Cloud Watch alarms and policies, which allow the size of the group to be scaled up or down. Typically, a maximum, a minimum, and a regular size for the group are specified.

An Auto Scaling group consists of a set of instances described in a static fashion by launch configurations. When the group scales up, new instances are started using the parameters for the run Instances EC2 call provided by the launch configuration. When the group scales down, the instances with older launch configurations are terminated first. The monitoring function of the Auto Scaling service carries out health checks to enforce the specified policies; for example, a user may specify a health check for elastic load balancing and then Auto Scaling will terminate an instance exhibiting a low performance and start a new one.

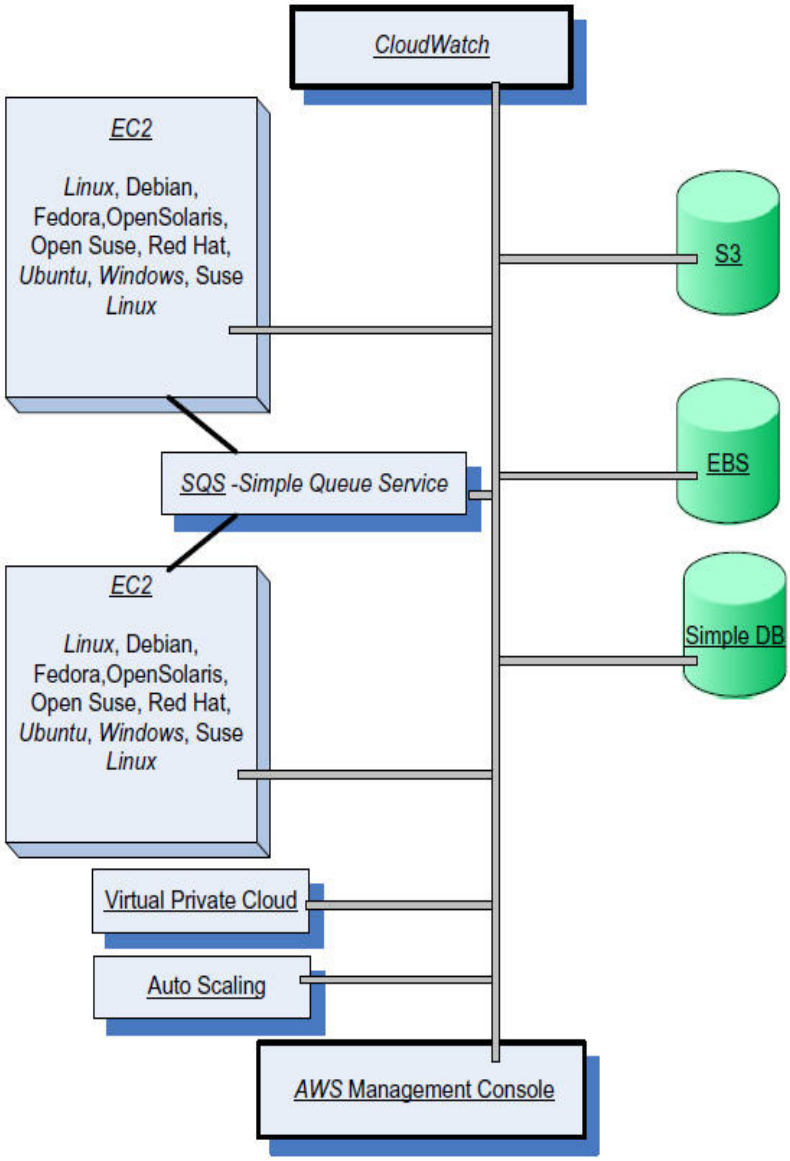


FIGURE 3.1

Regions and Availability Zones. Today Amazon offers cloud services through a network of datacenters on several continents, (see Table 3.15). In each region there are several availability zones interconnected by high-speed networks; regions communicate through the Internet and do not share resources.

An availability zone is a data center consisting of a large number of servers. A server may run multiple virtual machines or instances, started by one or more users; an instance may use storage services, S3, EBS), and Simple DB, as well as other services provided by AWS (see Figure 3.2). A cloud interconnect allows all systems in an availability zone to communicate with one another and with systems in other availability zones of the same region.

Storage is automatically replicated within a region; S3 buckets are replicated within an availability zone and between the availability zones of a region, whereas EBS volumes are replicated only within the same availability zone. Critical applications are advised to replicate important information in multiple regions to be able to function when the servers in one region are unavailable due to catastrophic events.

A user can request virtual servers and storage located in one of the regions. The user can also request virtual servers in one of the availability zones of that region. The Elastic Compute Cloud (EC2) service allows a user to interact and to manage the virtual servers.

The billing rates in each region to minimize costs, reduce communication latency, and increase reliability and security.

An instance is a virtual server. The user chooses the region and the availability zone where this virtual server should be placed and selects from a limited menu of instance types: the one that provides the resources, CPU cycles, main memory, and secondary storage, communication, and I/O bandwidth needed by the application.

When launched, an instance is provided with a DNS name. This name maps to a private IP address for internal communication within the internal EC2 communication network and a public IP address for communication outside the internal Amazon network, (e.g., for communication with the user that launched the instance). Network Address Translation (NAT) maps external IP addresses to internal ones. The public IP address is assigned for the lifetime of an instance and it is returned to the pool of available public IP addresses when the instance is either stopped or terminated. An instance can request an elastic IP address, rather than a public IP address. The elastic IP address is a static public IP address allocated to an instance from the available pool of

the availability zone. An elastic IP address is not released when the instance is stopped or terminated and must be released when no longer needed.

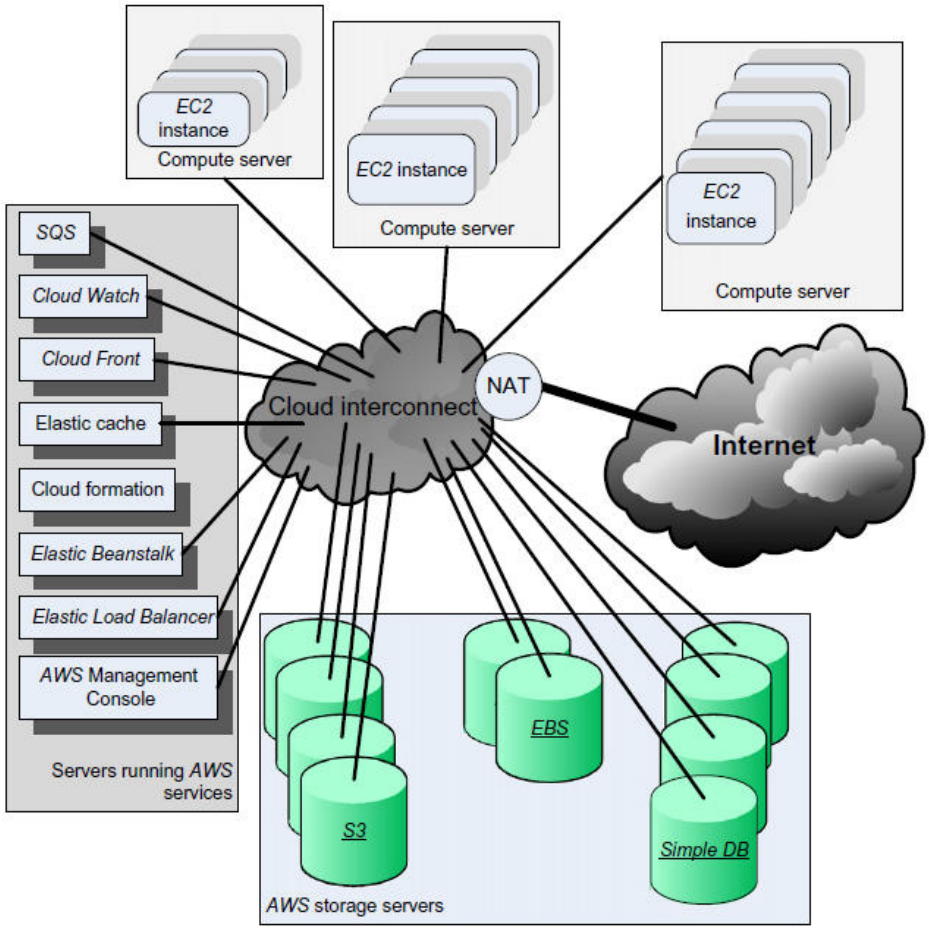


FIGURE 3.2

Table 3.1 Amazon data centers are located in several regions; in each region there are multiple availability zones. The billing rates differ from one region to another and can be roughly grouped into four categories: low, medium, high, and very high.

Region	Location	Availability Zones	Cost
US West	Oregon	us-west-2a/2b/2c	Low
US West	North California	us-west-1a/1b/1c	High
US East	North Virginia	us-east-1a/2a/3a/4a	Low
Europe	Ireland	eu-west-1a/1b/1c	Medium
South America	Sao Paulo, Brazil	sa-east-1a/1b	Very high
Asia/Pacific	Tokyo, Japan	ap-northeast-1a/1b	High
Asia/Pacific	Singapore	ap-southeast-1a/1b	Medium

The Charges for Amazon Web Services. Amazon charges a fee for EC2 instances, EBS storage, data transfer, and several other services. The charges differ from one region to another and depend on the pricing model; see http://aws.amazon.com/ec2/pricing for the current pricing structure.

There is three pricing models for EC2 instances: on-demand, reserved, and spot. On-demand instances use a flat hourly rate, and the user is charged for the time an instance is running; no reservation is required for this most popular model. For reserved instances a user pays a one-time fee to lock in a typically lower hourly rate. This model is advantageous when a user anticipates that the application will require a substantial number of CPU cycles and this amount is known in advance. Additional capacity is available at the larger standard rate. In case of spot instances, users bid on unused capacity and their instances are launched when the market price reaches a threshold specified by the user.

The EC2 system offers several instance types:

• Standard instances. Micro (StdM), small (StdS), large (StdL), extra large (StdXL); small is the default.

• High memory instances. High-memory extra-large (HmXL), high-memory double extra-large (Hm2XL), and high-memory quadruple extra-large (Hm4XL).

• High CPU instances. High-CPU extra-large (HcpuXL).

• Cluster computing. Cluster computing quadruple extra-large (Cl4XL).

Cloud computing: the Google perspective

Google's effort is concentrated in the area of Software-as-a-Service (SaaS). It is estimated that the number of servers used by Google was close to 1.8 million in January 2012 and was expected to reach close to 2.4 million in early 2013.

Services such as Gmail, Google Drive, Google Calendar, Picasa, and Google Groups are free of charge for individual users and available for a fee for organizations. These services are running on a cloud and can be invoked from a broad spectrum of devices, including mobile ones such as iPhones, iPads,Black-Berrys, and laptops and tablets. The data for these services is stored in data centers on the cloud.

The Gmail service hosts emails on Google servers and, provides a Web interface to access them and tools for migrating from Lotus Notes and Microsoft Exchange. Google Docs is Web-based software for building text documents, spreadsheets, and presentations. It supports features such

as tables, bullet points, basic fonts, and text size; it allows multiple users to edit and update the same document and view the history of document changes; and it provides a spell checker. The service allows users to import and export files in several formats, including Microsoft Office, PDF, text, and Open Office extensions.

Google Calendar is a browser-based scheduler; it supports multiple calendars for a user, the ability to share a calendar with other users, the display of daily/weekly/monthly views, and the ability to search events and synchronize with the Outlook Calendar. Google Calendar is accessible from mobile devices. Event reminders can be received via SMS, desktop popups, or emails. It is also possible to share your calendar with other Google Calendar users. Picasa is a tool to upload, share, and edit images; it provides 1 GB of disk space per user free of charge. Users can add tags to images and attach locations to photos using Google Maps. Google Groups allows users to host discussion forums to create messages online or via email.

Google is also a leader in the Platform-as-a-Service (PaaS) space. AppEngine is a developer platform hosted on the cloud. Initially it supported only Python, but support for Java was added later and detailed documentation for Java is available. The database for code development can be accessed with Google Query Language (GQL) with a SQL-like syntax.

The concept of structured data is important to Google's service strategy. The change of search Philosophy reflects the transition from unstructured Web content to structured data, data that contains additional information, such as the place where a photograph was taken, information about the singer of a digital recording of a song, the local services at a geographic location, and so on.

Google Co-op allows users to create customized search engines based on a set of facets or categories.

Google Base is a service allowing users to load structured data from different sources to a central repository that is a very large, self-describing, semi-structured, heterogeneous database. It is self-describing because each item follows a simple schema: (item type, attribute names). Few users are aware of this service. Google Base is accessed in response to keyword queries posed on Google.com, provided that there is relevant data in the database.

Google Drive is an online service for data storage that has been available since April 2012. It gives users 5 GB of free storage and charges $4 per month for 20 GB. It is available for PCs,

Mac Books, iPhones, iPads, and Android devices and allows organizations to purchase up to 16 TB of storage.

Microsoft Windows Azure and online services

Azure and Online Services are, respectively, PaaS and SaaS cloud platforms from Microsoft. Windows Azure is an operating system, SQL Azure is a cloud-based version of the SQL Server, andAzure AppFabric (formerly .NET Services) is a collection of services for cloud applications.

Windows Azure has three core components (see Figure 3.3): Compute, which provides a computation environment; Storage for scalable storage; and Fabric Controller, which deploys, manages, and monitors applications; it interconnects nodes consisting of servers, high-speed connections, and switches.

The Content Delivery Network (CDN) maintains cache copies of data to speed up computations. The Connect subsystem supports IP connections between the users and their applications running on Windows Azure. The API interface to Windows Azure is built on REST, HTTP, and XML. The platform includes five services: Live Services, SQL Azure, AppFabric, SharePoint, and Dynamics CRM. A client library and tools are also provided for developing cloud applications in Visual Studio.

The computations carried out by an application are implemented as one or more roles; an application typically runs multiple instances of a role. We can distinguish (i) Web role instances used to create Web applications; (ii) Worker role instances used to run Windows-based code; and (iii) VM role instances that run a user-provided Windows Server 2008 R2 image.

Scaling, load balancing, memory management, and reliability are ensured by a fabric controller, a distributed application replicated across a group of machines that owns all of the resources in its environment – computers, switches, load balancers – and it is aware of every Windows Azure application.

The fabric controller decides where new applications should run; it chooses the physical servers to optimize utilization using configuration information uploaded with each Windows Azure application. The configuration information is an XML-based description of how many Web role instances, how many Worker role instances, and what other resources the application needs. The fabric controller uses this configuration file to determine how many VMs to create.

Blobs, tables, queues, and drives are used as scalable storage. A blob contains binary data; a container consists of one or more blobs. Blobs can be up to a terabyte and they may have

associated metadata (e.g., the information about where a JPEG photograph was taken). Blobs allow a Windows Azure role instance to interact with persistent storage as though it were a local NTFS6 file system. Queues enable Web role instances to communicate asynchronously with Worker role instances.

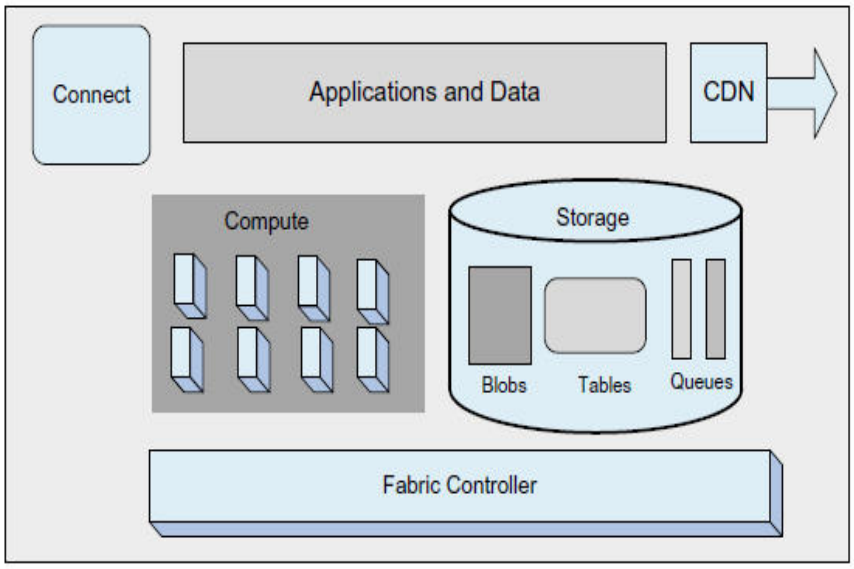


FIGURE 3.3

Open-source software platforms for private clouds

Private clouds provide a cost-effective alternative for very large organizations. A private cloud has essentially the same structural components as a commercial one: the servers, the network, virtual machines monitors (VMMs) running on individual systems, an archive containing disk images of virtual machines (VMs), a front end for communication with the user, and a cloud control infrastructure. Opensource cloud computing platforms such as Eucalyptus , Open Nebula, and Nimbus can be used as a control infrastructure for a private cloud.

Schematically, a cloud infrastructure carries out the following steps to run an application:

• Retrieves the user input from the front end.

• Retrieves the disk image of a VM from a repository.

• Locates a system and requests the VMM running on that system to set up a VM.

Invokes the DHCP7 and the IP bridging software to set up a MAC and IP address for the VM.

We discuss briefly the three open-source software systems, Eucalyptus, OpenNebula, and Nimbus.

Eucalyptus (www.eucalyptus.com) can be regarded as an open-source counterpart of Amazon's EC2,(see Figure 3.4). The systems support several operating systems including CentOS 5 and 6, RHEL 5 and 6, Ubuntu 10.04 LTS, and 12.04 LTS.

The components of the system are:

Virtual machine. Runs under several VMMs, including Xen, KVM, and Vmware.

• **Node controller**. Runs on every server or node designated to host a VM and controls the activities of the node.Reports to a cluster controller.

• **Cluster controller**. Controls a number of servers. Interacts with the node controller on each server to schedule requests on that node. Cluster controllers are managed by the cloud controller.

Cloud controller.Provides the cloud access to end users, developers, and administrators. It is accessible through command-line tools compatible withEC2 and through aWeb-based Dashboard. Manages cloud resources, makes high-level scheduling decisions, and interacts with cluster controllers.

• **Storage controller**.Provides persistent virtual hard drives to applications. It is the correspondent of EBS. Users can create snapshots from EBS volumes. Snapshots are stored in Walrus and made available across availability zones.

• **Storage service (Walrus).** Provides persistent storage and, similarly to S3, allows users to store objects in buckets.

The system supports a strong separation between the user space and the administrator space; users access the system via a Web interface, whereas administrators need root access. The system supports a decentralized resource management of multiple clusters with multiple cluster controllers, but a single head node for handling user interfaces. The procedure to construct a virtual machine is based on the generic one described in

The euca2ools front end is used to request a VM.

• The VM disk image is transferred to a compute node.

• This disk image is modified for use by the VMM on the compute node.

• The compute node sets up network bridging to provide a virtual network interface controller (NIC)8

with a virtual Media Access Control (MAC) address.9

• In the head node the DHCP is set up with the MAC/IP pair.

• VMM activates the VM.

• The user can now ssh10 directly into the VM.

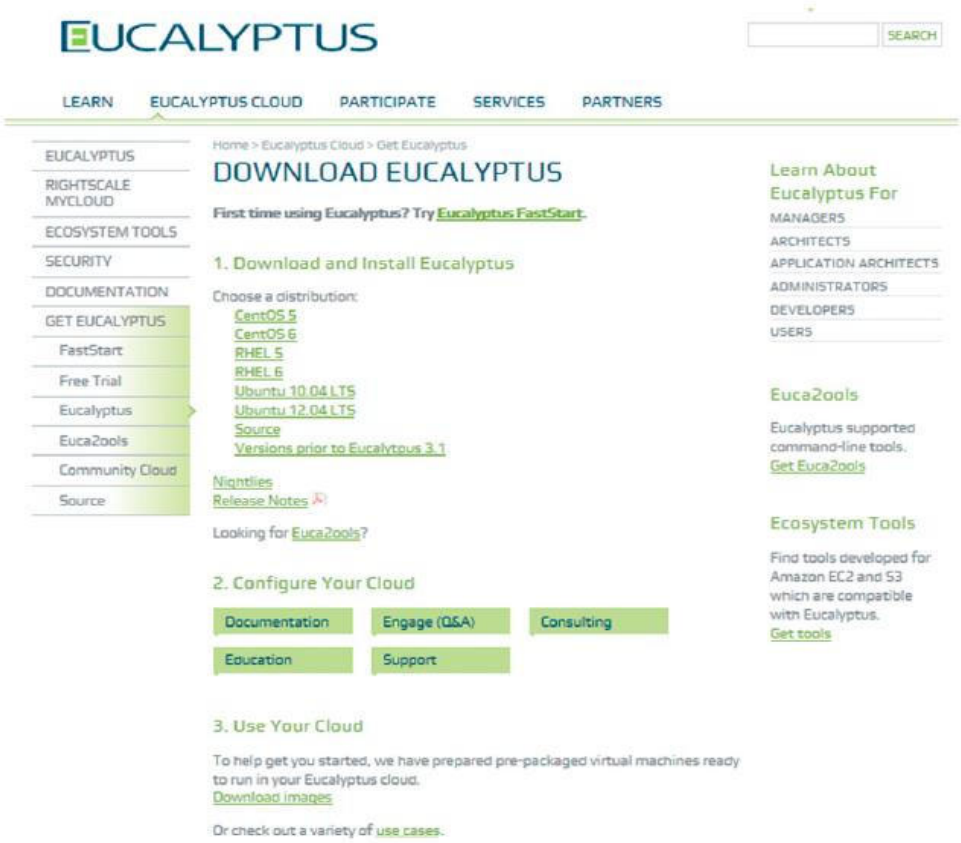


FIGURE 3.4

Eucalyptus supports several distributions and is well-documented software for private clouds.

Open-Nebula (www.opennebula.org) is a private cloud with users actually logging into the head node to access cloud functions. The system is centralized and its default configuration uses NFS (Network File System). The procedure to construct a virtual machine consists of several steps: (i) the user signs into the head node using ssh; (ii) the system uses the onevm command to request a VM; (iii) the VM template disk image is transformed to fit the correct size and configuration within the NFS directory on the head node; (iv) the oned daemon on the head node uses ssh to log into a compute node; (v) the compute node sets up network bridging to provide a virtual NIC with a virtual MAC; (vi) the files needed by the VMM are transferred to the compute node via the NFS; (vii) the VMM on the compute node starts the VM; and (viii) the user is able to ssh directly to the VM on the compute node.

Nimbus (www.nimbusproject.org) is a cloud solution for scientific applications based on the Globus software. The system inherits from Globus the image storage, the credentials for user authentication, and the requirement that a running Nimbus process can ssh into all compute

nodes. Customization in this system can only be done by the system administrators.

OpenStack is an open-source project started in 2009 at the National Aeronautics and Space Administration (NASA) in collaboration with Rackspace (www.rackspace.com) to develop a scalable cloud operating system for farms of servers using standard hardware. Though recently NASA has moved its cloud infrastructure to AWS in addition to Rackspace, several other companies, including HP, Cisco,IBM, and Red Hat, have an interest in Open Stack. The current version of the system supports a wide range of features such as application programming interfaces (APIs) with rate limiting and authentication; live VM management to run, reboot, suspend, and terminate instances; role-based access control; and the ability to allocate, track, and limit resource utilization. The administrators and the users control their resources using an extensible Web application called the Dashboard.

Table 3.5 A side-by-side comparison of *Eucalyptus*, *OpenNebula*, and *Nimbus*.

	Eucalyptus	*OpenNebula*	*Nimbus*
Design	Emulate *EC2*	Customizable	Based on Globus
Cloud type	Private	Private	Public/Private
User population	Large	Small	Large
Applications	All	All	Scientific
Customizability	Administrators and limited users	Administrators and users	All but image storage and credentials
Internal security	Strict	Loose	Strict
User access	User credentials	User credentials	x509 credentials
Network access	To cluster controller	—	To each compute node

Cloud storage diversity and vendor lock-in

There are several risks involved when a large organization relies solely on a single cloud provider. As the short history of cloud computing shows, cloud services may be unavailable for a short or even an extended period of time. Such an interruption of service is likely to negatively impact the organization and possibly diminish or cancel completely the benefits of utility computing for that organization. The potential for permanent data loss in case of a catastrophic system failure poses an equally great danger.

A solution to guarding against the problems posed by the vendor lock-in is to replicate the data to multiple cloud service providers. Straightforward replication is very costly and, at the same

time, poses technical challenges. The overhead to maintain data consistency could drastically affect the performance of the virtual storage system consisting of multiple full replicas of the organization's data spread over multiple vendors. Another solution could be based on an extension of the design principle of a RAID-5 system used for reliable data storage.

A RAID-5 system uses block-level stripping with distributed parity over a disk array, as shown in Figure 3.5(a); the disk controller distributes the sequential blocks of data to the physical disks and computes a parity block by bit-wise XOR-ing of the data blocks. The parity block is written on a different disk for each file to avoid the bottleneck possible when all parity blocks are written to a dedicated disk, as is done in the case of RAID-4 systems. This technique allows us to recover the data after a single disk loss. For example, if Disk 2 in Figure 3.5 is lost, we still have all the blocks of the third file, c1,c2, and c3, and we can recover the missing blocks for the others as follows:

$$\begin{aligned} a2 &= (a1)\ \text{XOR}\ (aP)\ \text{XOR}\ (a3) \\ b2 &= (b1)\ \text{XOR}\ (bP)\ \text{XOR}\ (b3)\ . \\ d1 &= (dP)\ \text{XOR}\ (d2)\ \text{XOR}\ (d3) \end{aligned}$$

Obviously, we can also detect and correct errors in a single block using the same procedure. The RAID controller also allows parallel access to data (for example, the blocks a1, a2, and a3 can be read and written concurrently) and it can aggregate multiple write operations to improve performance.

The system in Figure 3.5(b) strips the data across four clusters. The access to data is controlled by a proxy that carries out some of the functions of a RAID controller, as well as authentication and other security-related functions. The proxy ensures before-and-after atomicity as well as all-or-nothing atomicity for data access; the proxy buffers the data, possibly converts the data manipulation commands optimizes the data access (e.g., aggregates multiple write operations), converts data to formats specific to each cloud, and so on.

The Redundant Array of Cloud Storage (RACS) system uses the same data model and mimics the interface of the S3 provided by AWS.The S3 system, discussed in Section 3.1, stores the data in buckets, each bucket being a flat namespace with keys associated with objects of arbitrary size but less than 5 GB. The prototype implementation discussed in [5] led the authors to conclude that the cost increases and the performance penalties of the RACS systems are relatively minor.

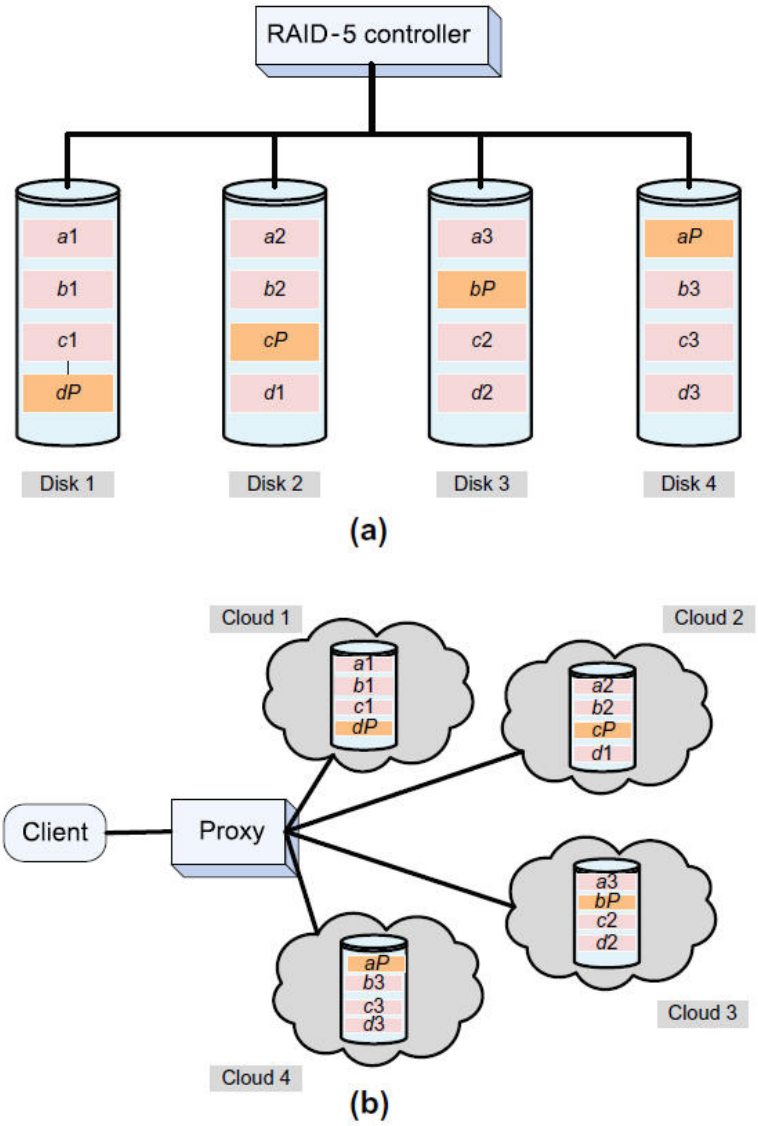


FIGURE 3.5

Cloud computing interoperability: the Intercloud

Cloud interoperability could alleviate the concern that users could become hopelessly dependent on a single cloud service provider, the so-called vendor lock-in.It seems natural to ask the question whether an Intercloud – a "cloud of clouds," a federation of clouds that cooperate to provide a better user experience – is technically and economically feasible. The Internet is a network of networks; hence, it appears that an Intercloud seems plausible.

Closer scrutiny shows that the extension of the concept of interoperability from networks to clouds is far from trivial. A network offers one high-level service, the transport of digital information from a source, a host outside a network, to a destination, another host, or another network that can deliver the information to its final destination. This transport of information through a network of networks is feasible because before the Internet was born, agreements on basic questions were reached.

The situation is quite different in cloud computing. First, there are no standards for storage of processing; second, the clouds we have seen so far are based on different delivery models: SaaS, PaaS,and IaaS. Moreover, the set of services supported by each of these delivery models is not only large, it is open; new services are offered every few months. For example, in October 2012 Amazon announced a new service, the AWS GovCloud (US).

The question of whether cloud service providers (CSPs) are willing to cooperate to build an Inter cloud is open. Some CSPs may think that they have a competitive advantage due to the uniqueness of the added value of their services. Thus, exposing how they store and process information may adversely affect their business. Moreover, no CSP will be willing to change its internal operation, so a first question is whether an Inter cloud could be built under these c

An Intercloud would then require the development of an ontology11 for cloud computing. Then each cloud service provider would have to create a description of all resources and services using this ontology. Due to the very large number of systems and services, the volume of information provided by individual cloud service providers would be so large that a distributed database not unlike the Domain Name Service (DNS) would have to be created and maintained.

Each cloud would then require an interface, a so-called Intercloud exchange, to translate the common language describing all objects and actions included in a request originating from another cloud in terms of its internal objects and actions. To be more precise, a request originated in one cloud would have to be translated from the internal representation in that cloud to a common representation based on the shared ontology and then, at the destination, it would be translated into an internal representation that can be acted on by the destination cloud.

The Public Key Infrastructure (PKI),12 an all-or-nothing trust model, is not adequate for an Intercloud, where the trust must be nuanced. A nuanced model for handling digital certificates means that one cloud acting on behalf of a user may grant access to another cloud to read data in storage, but not to start new instances.

Encryption must be used to protect the data in storage and in transit in the Intercloud. The OASIS13 Key Management Interoperability Protocol (KMIP)14 is proposed for key management.

Energy use and ecological impact of large-scale data centers

We start our discussion of energy use by data centers and its economic and ecological impact with a brief analysis of the concept of energy-proportional systems. This is a very important concept because a strategy for resource management in a computing cloud is to concentrate the load on a subset of servers and switching the rest of the servers to a standby mode whenever possible.

The operating efficiency of a system is captured by an expression of "performance per Watt of power." It is widely reported that, during the last two decades, the performance of computing systems has increased much faster than their operating efficiency.

Energy-proportional systems could lead to large savings in energy costs for computing clouds. An energy-proportional system consumes no power when idle, very little power under a light load, and gradually more power as the load increases. By definition, an ideal energy-proportional system is always operating at 100% efficiency. Humans are a good approximation of an ideal energy proportional system; human energy consumption is about 70 W at rest and 120 W on average on a daily basis and can go as high as 1,000–2,000 W during a strenuous, short effort.

A number of proposals have emerged for energy-proportional networks; the energy consumed by such networks is proportional to the communication load.

High-speed channels typically consist of multiple serial lanes with the same data rate; a physical unit is stripped across all the active lanes. Channels commonly operate plesiochronously16 and are always on, regardless of the load, because they must still send idle packets to maintain byte and lane alignment across the multiple lanes.

Energy saving in large-scale storage systems is also of concern. A strategy to reduce energy consumption is to concentrate the workload on a small number of disks and allow the others to operate in a low-power mode. One of the techniques to accomplish this task is based on replication. A replication strategy based on a sliding window is reported in [364]; measurement results indicate that it performs better than LRU, MRU, and LFU17 policies for a range of file sizes, file availability, and number of client nodes, and the power requirements are reduced by as much as 31%.

Another technique is based on data migration. The system in [158] uses data storage in virtual nodes managed with a distributed hash table; the migration is controlled by two algorithms, a short-term optimization algorithm, used for gathering or spreading virtual nodes according to the daily variation of the workload so that the number of active physical nodes is reduced to a minimum, and a long-term optimization algorithm, used for coping with changes in the popularity of data over a longer period.

The energy consumption of large-scale data centers and their costs for energy and for cooling are significant now and are expected to increase substantially in the future. In 2006, the 6,000 data centers in the United States reportedly consumed 61×109 KWh of energy, 1.5% of all electricity consumption in the country, at a cost of $4.5 billion [364].

The effort to reduce energy use is focused on the computing, networking, and storage activities of a data center.

Many proposals argue that dynamic resource provisioning is necessary to minimize power consumption. Two main issues are critical for energy saving: the amount of resources allocated to each application and the placement of individual workloads. For example, a resource management framework combining a utility-based dynamic virtual machine provisioning manager with a dynamic VM placement manager to minimize power consumption and reduce SLA violations is presented in.

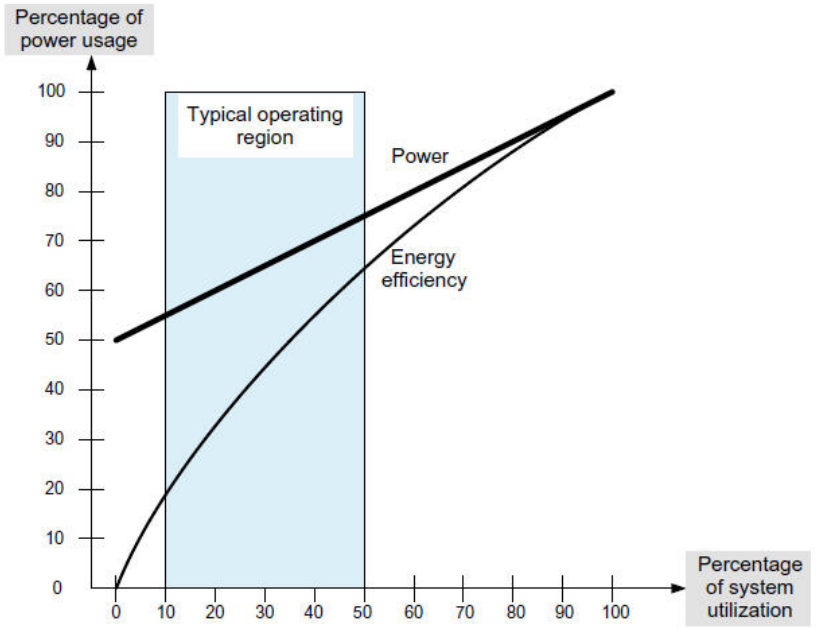


FIGURE 3.6

Service- and compliance-level agreements

A service-level agreement (SLA) is a negotiated contract between two parties, the customer and the service provider. The agreement can be legally binding or informal and specifies the services that the customer receives rather than how the service provider delivers the services. The objectives of the agreement are.

Identify and define customers' needs and constraints, including the level of resources, security, timing, and quality of service.

• Provide a framework for understanding. A critical aspect of this framework is a clear definition of classes of service and costs.

• Simplify complex issues; for example, clarify the boundaries between the responsibilities of the clients and those of the provider of service in case of failures.

• Reduce areas of conflict.

• Encourage dialogue in the event of disputes.

• Eliminate unrealistic expectations

An SLA records a common understanding in several areas: (i) services, (ii) priorities, (iii) responsibilities, (iv) guarantees, and (v) warranties. An agreement usually covers: services to be delivered, performance, tracking and reporting, problem management, legal compliance and resolution of disputes, customer duties and responsibilities, security, handling of confidential information, and termination.

Each area of service in cloud computing should define a "target level of service" or a "minimum level of service" and specify the levels of availability, serviceability, performance, operation, or other attributes of the service, such as billing. Penalties may also be specified in the case of noncompliance with the SLA. It is expected that any service-oriented architecture (SOA) will eventually include middleware supporting SLA management.

There are two well-differentiated phases in SLA management: the negotiation of the contract and the monitoring of its fulfillment in real time. In turn, automated negotiation has three main components: i) the object of negotiation, which defines the attributes and constraints under negotiation; (ii) the negotiation protocols, which describe the interaction between negotiating parties; and (iii) the decision models responsible for processing proposals and generating counterproposals.

The selection process is subject to customizable compliance with user requirements, such as security, deadlines, and costs. The authors propose an infrastructure called Compliant Cloud Computing (C3) consisting of: (i) a language to express user requirements and the compliance level agreements (CLAs) and (ii) the middleware for managing CLAs.

The Web Service Agreement Specification (WS-Agreement) [20] uses an XML-based language to define a protocol for creating an agreement using a predefined template with some customizable aspects. It only supports one-round negotiation without counterproposals.

Responsibility sharing between user and cloud service provider

After reviewing cloud services provided by Amazon, Google, and Microsoft, we are in a better position to understand the differences among SaaS, IaaS, and PaaS. There is no confusion about SaaS;the service provider supplies both the hardware and the application software, and the user has direct access to these services through a Web interface and has no control over cloud resources. Typical examples are Google with Gmail, Google Docs, Google Calendar, Google Groups, and Picasa and Microsoft with the Online Services.

In the case of IaaS, the service provider supplies the hardware (servers, storage, networks) and system software (operating systems, databases); in addition, the provider ensures system attributes such as security, fault tolerance, and load balancing. The representative of IaaS is Amazon AWS.

PaaS provides only a platform, including the hardware and system software, such as operating systems and databases; the service provider is responsible for system updates, patches, and software maintenance.

PaaS does not allow any user control of the operating system, security features, or the ability to install applications. Typical examples are Google App Engine, Microsoft Azure, and Force.com, provided by Salesforce.com.

The level of user control over the system in IaaS is different form PaaS. IaaS provides total control, whereas PaaS typically provides no control. Consequently, IaaS incurs administration costs similar to a traditional computing infrastructure, whereas the administrative costs are virtually zero for PaaS.

The limits of responsibility between the cloud user and the cloud service provider are different for the three service-delivery models, as we can see in Figure 3.7. In the case of SaaS the user is partially responsible for the interface; the user responsibility increases in the case of PaaS and

includes the interface and the application. In the case of IaaS the user is responsible for all the events occurring in the virtual machine running the application.

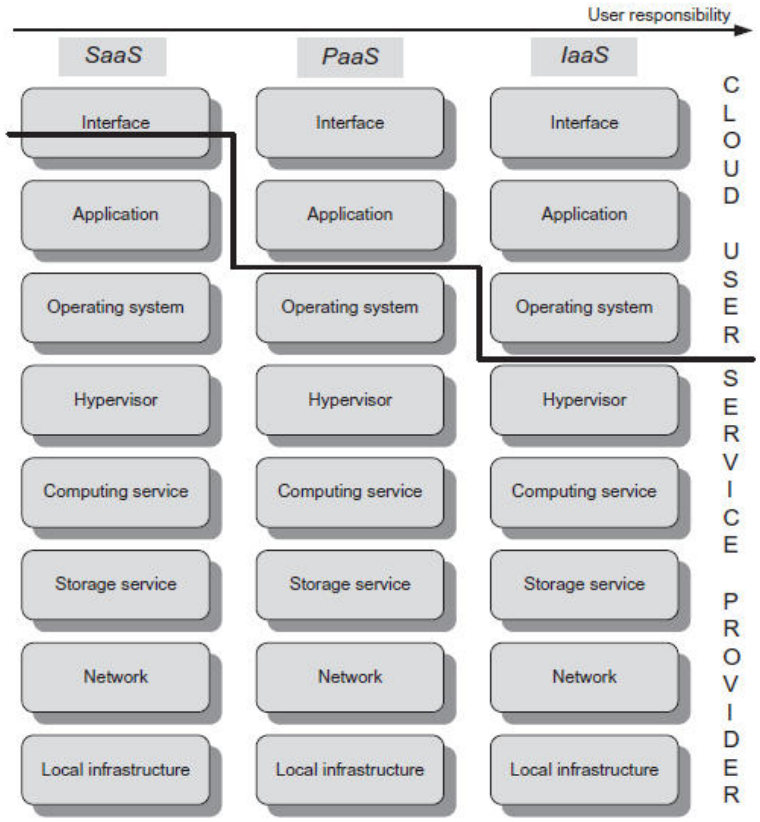


FIGURE 3.7

The limits of responsibility between a cloud user and the cloud service provider.

User experience

There have been a few studies of user experience based on a large population of cloud computing Users. An empirical study of the experience of a small group of users of the Finish Cloud Computing Consortium is reported in.

The security threats perceived by this group of users are: (i) abuse and villainous use of the cloud;(ii) APIs that are not fully secure; (iii) malicious insiders; (iv) account hijacking; (iv) data leaks; and (v) issues related to shared resources. Identity theft and privacy were major concerns for about half of the users questioned; availability, liability, and data ownership and copyright were raised by a third of respondents.

The suggested solutions to these problems are as follows: SLAs and tools to monitor usage should be deployed to prevent abuse of the cloud; data encryption and security testing should

enhance the API security; an independent security layer should be added to prevent threats caused by malicious insiders; strong authentication and authorization should be enforced to prevent account hijacking; data decryption in a secure environment should be implemented to prevent data leakage; and compartmentalization of components and firewalls should be deployed to limit the negative effect of resource sharing.

A broad set of concerns identified by the NIST working group on cloud security includes:

. Potential loss of control/ownership of data.

• Data integration, privacy enforcement, data encryption.

• Data remanence after deprovisioning.

• Multitenant data isolation.

• Data location requirements within national borders.

• Hypervisor security.

• Audit data integrity protection.

• Verification of subscriber policies through provider controls.

• Certification/accreditation requirements for a given cloud service.

Software Licensing

Software licensing for cloud computing is an enduring problem without a universally accepted solution at this time. The license management technology is based on the old model of computing centers with licenses given on the basis of named users or as site licenses. This licensing technology, developed for a centrally managed environment, cannot accommodate the distributed service infrastructure of cloud computing or of grid computing.

Only very recently IBM reached an agreement allowing some of its software products to be used on EC2. Furthermore, MathWorks developed a business model for the use ofMATLAB in grid environments. The Software-as-a-Service (SaaS) deployment model is gaining acceptance because it allows users to pay only for the services they use.

There is significant pressure to change the traditional software licensing model and find non hardware-based solutions for cloud computing. The increased negotiating power of users, coupled with the increase in software piracy, has renewed interest in alternative schemes such as those proposed by the SmartLM research project (www.smartlm.eu). SmartLM license management requires a complex software infrastructure involving SLA, negotiation protocols, authentication, and other management functions.

A commercial product based on the ideas developed by this research project is elasticLM, which provides license and billing for Web-based services. The architecture of the elasticLM license service has several layers: allocation, authentication, administration, management, business, and persistency. The authentication layer authenticates communication between the license service and the billing service as well as the individual applications; the persistence layer stores the usage records. The main responsibility of the business layer is to provide the licensing service with the licenses prices, and the management coordinates various components of the automated billing service.

When a user requests a license from the license service, the terms of the license usage are negotiated and they are part of an SLA document. The negotiation is based on application-specific templates and the license cost becomes part of the SLA. The SLA describes all aspects of resource usage, including the ID of application, duration, number of processors, and guarantees, such as the maximum cost and deadlines. When multiple negotiation steps are necessary, the WS-Agreement Negotiation protocol is used.

ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES
RAJAMPET
(Autonomous)

Department of Artificial Intelligence & Machine Learning

Lecture Notes

Name of the Faculty: **Dr.T.Harikrishna** **Class:** **IVYear I Semester**

Branch and Section: **AIML** **Course code:** **20A57GT**

Name of the Course: Cloud Computing

Unit-II

Unit-II

Cloud Computing: Applications and Paradigms

Cloud computing is very attractive to users for several economic reasons: It requires a very low infrastructure investment because there is no need to assemble and maintain a large-scale system and it has low utility-based computing costs because customers are only billed for the infrastructure used. At the same time, users benefit from the potential to reduce the execution time of compute-intensive and data-intensive applications through parallelization. If an application can partition the workload in n segments and spawn n instances of itself, the execution time could be reduced by a factor close to n.

Moreover, because application developers enjoy the advantages of a just-in-time infrastructure, they are free to design an application without being concerned with the system where the application will run. Often, an application becomes the victim of its own success, attracting a user population larger than the system can support. Cloud elasticity allows such an application to absorb the additional workload without any effort from application developers.

Challenges for cloud computing

One of the main advantages of cloud computing, the shared infrastructure, could also have a negative impact. Performance isolation is nearly impossible to reach in a real system, especially when the system is heavily loaded. The performance of virtual machines fluctuates based on the load, the infrastructure services, and the environment, including the other users. Security isolation is also challenging on multitenant systems.

Reliability is also a major concern; node failures are to be expected whenever a large number of nodes cooperate for the computations. Choosing an optimal instance (interms of performance isolation, reliability, and security) from those offered by the cloud infrastructure is another critical factor to be considered. Of course, cost considerations also play a role in the choice of the instance type.

Many applications consist of multiple stages; in turn, each stage may involve multiple instances running in parallel on the systems of the cloud and communicating among them. Thus, efficiency, consistency, and communication scalability are major concerns for an application developer.

Data storage plays a critical role in the performance of any data-intensive application; the organization of the storage, the storage location, and the storage bandwidth must be carefully analysed to lead to optimal application performance. Clouds support many storage options to

set up a file system similar to the Hadoop file system among them are off-instance cloud storage (e.g., S3), mountable off-instance block storage (e.g., EBS), and storage persistent for the lifetime of the instance.

Many data-intensive applications use metadata associated with individual data records

Another important consideration for the application developer is logging. Performance considerations limit the amount of data logging, whereas the ability to identify the source of unexpected results and errors is helped by frequent logging. Logging is typically done using instance storage preserved only for the lifetime of the instance.

Existing cloud applications and new application opportunities

Existing cloud applications can be divided into several broad categories: (i) processing pipelines; (ii) batch processing systems; and (iii) Web applications.

Processing pipelines are data-intensive and sometimes compute-intensive applications and represent a fairly large segment of applications currently running on the cloud. Several types of data processing applications can be identified.

Processing pipelines are data-intensive and sometimes compute-intensive applications and represent a fairly large segment of applications currently running on the cloud. Several types of data processing applications can be identified:

Indexing. The processing pipeline supports indexing of large datasets created by Web crawler engines.

Data mining. The processing pipeline supports searching very large collections of records to locate items of interests.

Image processing .A number of companies allow users to store their images on the cloud The image-processing pipelines support image conversion (e.g. Enlarging an image or creating thumbnails).They can also be used to compress or encrypt images.

Video transcoding. The processing pipeline transcodes from one video format to another (e.g.,from AVI to MPEG).

Document processing. The processing pipeline converts very large collections of documents from one format to another (e.g. From Word to PDF),or encrypts the documents. It could also use optical character recognition (OCR) to produce digital images of documents

Batch processing systems also cover a broad spectrum of data-intensive applications in enterprise computing. Such applications typically have deadlines, and the failure to meet these deadlines could have serious economic consequences. Security is also a critical aspect for many applications of batch processing. A non-exhaustive list of batch processing applications includes.

Generation of daily, weekly, monthly, and annual activity reports for organizations in retail, manufacturing, and other economic sectors.

• Processing, aggregation, and summaries of daily transactions for financial institutions, insurance companies, and healthcare organizations.

• Inventory management for large corporations.

• Processing billing and payroll records.

• Management of the software development (e.g., nightly updates of software repositories).

• Automatic testing and verification of software and hardware systems

Finally, and of increasing importance, are cloud applications in the area of Web access. Several categories of Web sites have a periodic or a temporary presence, such as the Web sites for conferences or other events. There are also Web sites that are active during a particular season. or that support a particular type of activity, such as income tax reporting with the April 15 deadline each year. Other limited-time Web sites used for promotional activities “sleep” during the night and auto-scale during the day.

Architectural styles for cloud applications

The vast majority of cloud applications take advantage of request/response communication between clients and stateless servers. A stateless server does not require a client to first establish a connection to the server. Instead, it views a client request as an independent transaction and responds to it.

The advantages of stateless servers are obvious. Recovering from a server failure requires considerable overhead for a server that maintains the state of all its connections, whereas in the case of stateless server a client is not affected while a server goes down and then comes back up between two consecutive requests. A stateless system is simpler, more robust, and scalable. A client does not have to be concerned with the state of the server. If the client receives a response to a request, that means that the server is up and running; if not, it should resend the request later. A connection-based service must reserve spaces to maintain the state of each connection with a client; therefore, such a system is not scalable, and the number of clients a server could interact with at any given time is limited by the storage space available to the server.

A critical aspect of the development of networked applications is how processes and threads running on systems with different architectures and possibly compiled from different programming languages can communicate structured information with one another. First, the

internal representation of the two structures at the two sites may be different. One system may use Big-Endian and the other Little-Endian representation. The character representations may also be different. Second, a communication channel transmits a sequence of bits and bytes; thus, the data structure must be serialized at the sending site and reconstructed at the receiving site.

Several other considerations must be analyzed before deciding on the architectural style of an application. The term neutrality refers to the ability of the application protocol to use different transport protocols such as TCP or UDP and, in general, to run on top of a different protocol stack..

RPC-based applications use stubs to convert the parameters involved in an RPC call. A stub performs two functions: marshalling the data structures and serialization. A more general concept is that of an Object Request Broker (ORB), the middleware that facilitates communication of networked applications. The ORB at the sending site transforms the data structures used internally by a sending process to a byte sequence and transmits this byte sequence over the network. The ORB at the receiving site maps the byte sequence to the data structures used internally by the receiving process.

The Common Object Request Broker Architecture (CORBA) was developed in the early 1990s to allow networked applications developed in different programming languages and running on systems with different architectures and system software to work with one another. At the heart of the system is the Interface Definition Language (IDL), used to specify the interface of an object.

The Simple Object Access Protocol (SOAP) is an application protocol developed in 1998 for Web applications; its message format is based on the Extensible Markup Language (XML). SOAP uses TCP and, more recently, UDP transport protocols. It can also be stacked above other application layer protocols such as HTTP, SMTP, or JMS. The processing model of SOAP is based on a network consisting of senders, receivers, intermediaries, message originators, ultimate receivers, and message paths. SOAP is an underlying layer of Web Services.

The Web Services Description Language (WSDL) (see www.w3.org/TR/wsdl) was introduced in 2001 as an XML-based grammar to describe communication between endpoints of a networked application. The abstract definition of the elements involved include services, collections of endpoints of communication; types, containers for data type definitions; operations, descriptions of actions supported by a service; port types, operations supported by

endpoints; bindings, protocols and data formats supported by a particular port type; and port, an endpoint as a combination of a binding and a network address.

Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems. REST supports client communication with stateless servers. It is platform- and languageindependent, supports data caching, and can be used in the presence of firewalls.

REST almost always uses HTTP to support all four Create/Read/Update/Delete (CRUD) operations. It uses GET, PUT, and DELETE to read, write, and delete the data, respectively. REST is a much easier-to-use alternative to RPC, CORBA, or Web Services such as SOAP or WSDL.

Workflows: Coordination of multiple activities

Many cloud applications require the completion of multiple interdependent tasks; the description of a complex activity involving such an ensemble of tasks is known as a workflow.

Workflow models are abstractions revealing the most important properties of the entities participating in a workflow management system. Task is the central concept in workflow modeling; a task is a unit of work to be performed on the cloud, and it is characterized by several attributes, such as.

Name. A string of characters uniquely identifying the task.

• **Description**. A natural language description of the task

Actions. Modifications of the environment caused by the execution of the task.

• **Preconditions.** Boolean expressions that must be true before the action(s) of the task can take place.

• **Post-conditions**. Boolean expressions that must be true after the action(s) of the task take place.

• **Attributes**. Provide indications of the type and quantity of resources necessary for the execution of the task, the actors in charge of the tasks, the security requirements, whether the task is reversible, and other task characteristics.

Exceptions. Provide information on how to handle abnormal events. The exceptions supported by a task consist of a list of <event, action> pairs. The exceptions included in the task exception list are called anticipated exceptions, as opposed to unanticipated exceptions.

A composite task is a structure describing a subset of tasks and the order of their execution. A primitive task is one that cannot be decomposed into simpler tasks. **A composite task**

inherits some properties from workflows; it consists of tasks and has one start symbol and possibly several end symbols. At the same time, a composite task inherits some properties from tasks; it has a name, preconditions, and post-conditions.

A routing task is a special-purpose task connecting two tasks in a workflow description. The task that has just completed execution is called the predecessor task; the one to be initiated next is called the successor task. A routing task could trigger a sequential, concurrent, or iterative execution. Several types of routing task exist.

A fork routing task triggers execution of several successor tasks. Several semantics for this construct are possible:

• All successor tasks are enabled.

• Each successor task is associated with a condition. The conditions for all tasks are evaluated, and only the tasks with a true condition are enabled.

• Each successor task is associated with a condition. The conditions for all tasks are evaluated, but the conditions are mutually exclusive and only one condition may be true. Thus, only one task is enabled.

• Nondeterministic, k out of n > k successors are selected at random to be enabled.

A join routing task waits for completion of its predecessor tasks. There are several semantics for the join routing task:

• The successor is enabled after all predecessors end.

• The successor is enabled after k out of n > k predecessors end.

• Iterative: The tasks between the fork and the join are executed repeatedly.

A process description, also called a workflow schema, is a structure describing the tasks or activities to be executed and the order of their execution. A process description contains one start symbol and one end symbol. A process description can be provided in a workflow definition language (WFDL),supporting constructs for choice, concurrent execution, the classical fork, join constructs, and iterative execution.

The phases in the life cycle of a workflow are creation, definition, verification, and enactment. There is a striking similarity between the life cycle of a workflow and that of a traditional computer program, namely, creation, compilation, and execution (see Figure 4.1). The workflow specification by means of a workflow description language is analogous to writing a program. Planning is equivalent to automatic program generation. Workflow verification corresponds to syntactic verification of a program, and workflow enactment mirrors the execution of a compiled program.

A case is an instance of a process description. The start and stop symbols in the workflow description enable the creation and the termination of a case, respectively. An enactment model describes the steps taken to process a case. When a computer executes all tasks required by a workflow the enactment can be performed by a program called an enactment engine.

An alternative description of a workflow can be provided by a transition system describing the possible paths from the current state to a goal state. Sometimes, instead of providing a process description, we may specify only the goal state and expect the system to generate a workflow description that could lead to that state through a set of actions. In this case, the new workflow description is generated automatically, knowing a set of tasks and the preconditions and post-conditions for each one of them. In artificial intelligence (AI) this activity is known as planning.

The state space of a process includes one initial state and one goal state; a transition system identifies all possible paths from the initial to the goal state. A case corresponds to a particular path in the transition system. The state of a case tracks the progress made during the enactment of that case.

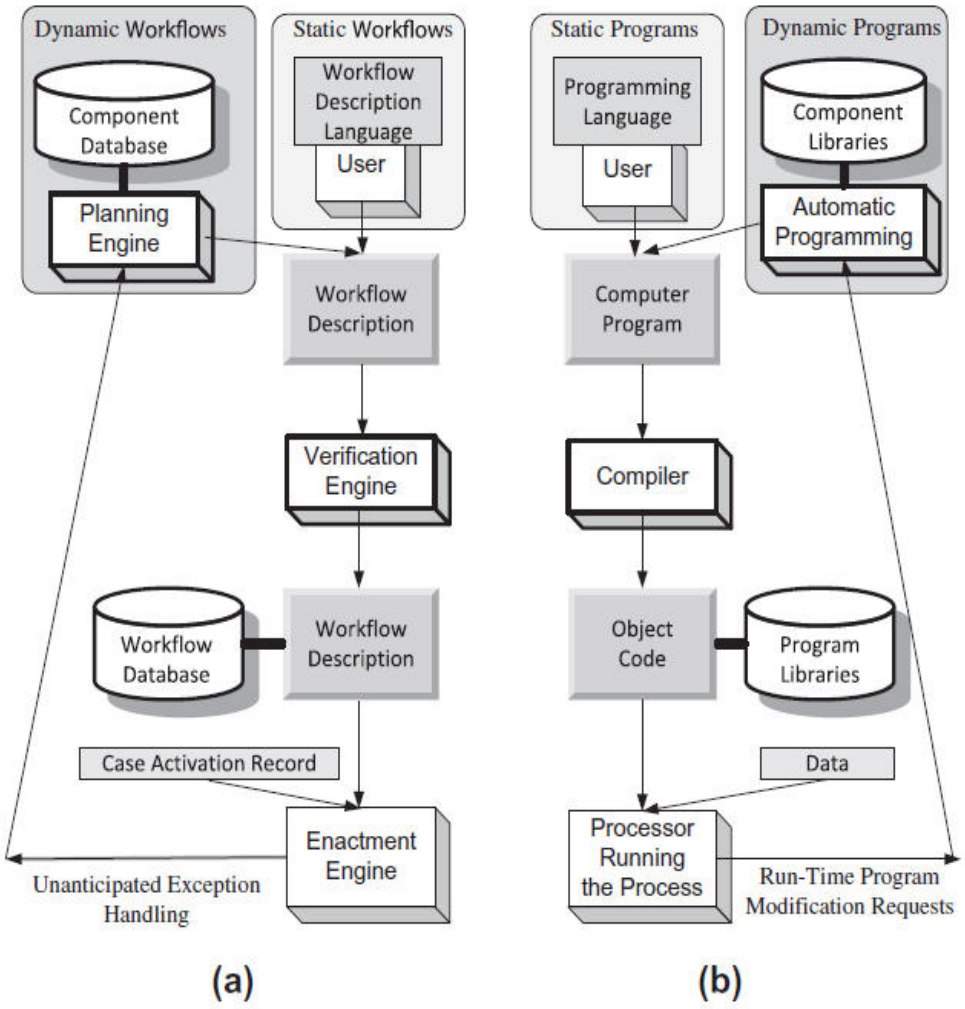


FIGURE 4.1

A parallel between workflows and programs. (a) The life cycle of a workflow. (b) The life cycle of a computer program. The workflow definition is analogous to writing a program. Planning is analogous to automatic program generation. Verification corresponds to syntactic verification of a program. Workflow enactment mirrors the execution of a program. A static workflow corresponds to a static program and a dynamic workflow to a dynamic program.

The state space of a process includes one initial state and one goal state; a transition system identifies all possible paths from the initial to the goal state. A case corresponds to a particular path in the transition system. The state of a case tracks the progress made during the enactment of that case.

For example, the process description in Figure 4.2(a) violates the liveness requirement.

As long as task C is chosen after completion of B, the process will terminate. However, if D is chosen, then F will never be instantiated, because it requires the completion of both C and E. The process will never terminate, because G requires completion of both D and F.

A note of caution: Although the original description of a process could be live, the actual enactment of a case may be affected by deadlocks due to resource allocation. To illustrate this situation, consider two tasks, A and B, running concurrently. Each of them needs exclusive access to resources r and q for a period of time. Either of two scenarios is possible.

1. A or B acquires both resources and then releases them and allows the other task to do the same.
2. We face the undesirable situation in Figure 4.2(b) when, at time t1, task A acquires r and continues its execution; then at time t2 task B acquires q and continues to run. Then at time t3 task B attempts to acquire r and it blocks because r is under the control of A. Task A continues to run and at time t4 attempts to acquire q and it blocks because q is under the control of B.

The deadlock illustrated in Figure 4.2(b) can be avoided by requesting each task to acquire all resources at the same time. The price to pay is underutilization of resources. Indeed, the idle time of each resource increases under this scheme.

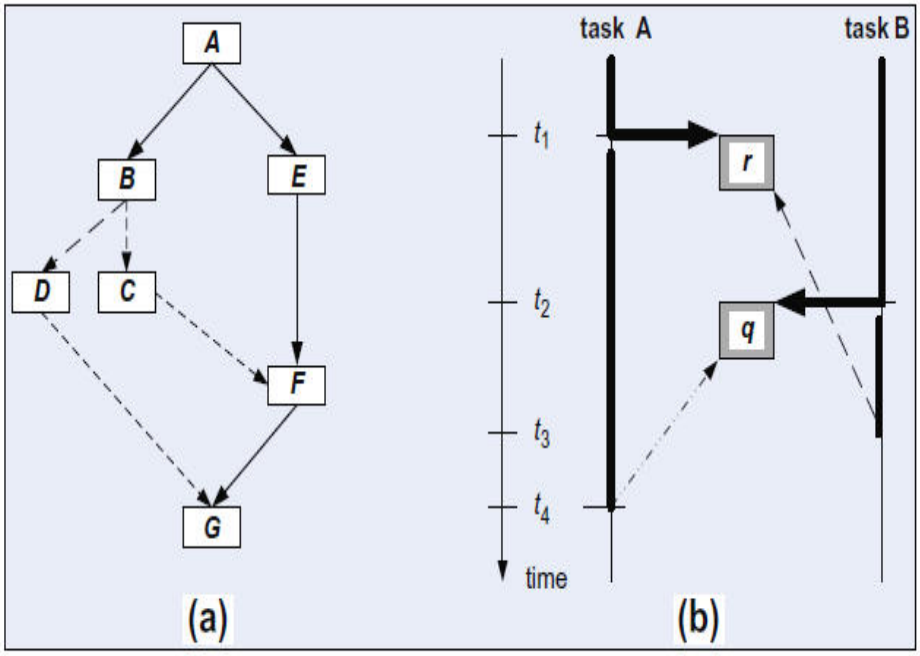


FIGURE 4.2

Workflow pattern refers to the temporal relationship among the tasks of a process. The workflow description languages and the mechanisms to control the enactment of a case must have provisions to support these temporal relationships. Workflow patterns are analyzed in [1,382]. These patterns

are classified in several categories: basic, advanced branching and synchronization, structural, statebased, cancellation, and patterns involving multiple instances. The basic workflow patterns illustrated in Figure 4.3 are

The sequence pattern occurs when several tasks have to be scheduled one after the completion of the other [see Figure 4.3(a)].

• **The AND split pattern** requires several tasks to be executed concurrently. Both tasks B and C are activated when task A terminates [see Figure 4.3(b)]. In case of an explicit AND split, the activity graph has a routing node and all activities connected to the routing node are activated as soon as the flow of control reaches the routing node. In the case of an implicit AND split, activities are connected directly and conditions can be associated with branches linking an activity with the next ones. Only when the conditions associated with a branch are true are the tasks activated.

The synchronization pattern requires several concurrent activities to terminate before an activity can start. In our example, task C can only start after both tasks A and B terminate

The XOR split requires a decision; after the completion of task A, either B or C can be activated

In the XOR join, several alternatives are merged into one. In our example, task C is enabled when either A or B terminates.

The OR split pattern is a construct to choose multiple alternatives out of a set. In our example, after completion of task A, one could activate either B or C, or both.

The multiple merge construct allows multiple activations of a task and does not require synchronization after the execution of concurrent tasks. Once A terminates, tasks B and C execute concurrently. When the first of them, say, B, terminates, task D is activated; then when C terminates, D is activated again.

The discriminator pattern waits for a number of incoming branches to complete before activating the subsequent activity [see Figure 4.3(h)]; then it waits for the remaining branches to finish without taking any action until all of them have terminated. Next, it resets itself.

The N out of M join construct provides a barrier synchronization. Assuming that M > N tasks run concurrently, N of them have to reach the barrier before the next task is enabled. In our example, any two out of the three tasks A, B, and C have to finish before E is enabled.

The deferred choice pattern is similar to the XOR split, but this time the choice is not made explicitly and the run-time environment decides what branch to take

The deferred choice pattern is similar to the XOR split, but this time the choice is not made explicitly and the run-time environment decides what branch to take.

Some workflows are static. The activity graph does not change during the enactment of a case. Dynamic workflows are those that allow the activity graph to be modified during the enactment of a case. Some of the more difficult questions encountered in dynamic workflow management refer to (i) how to integrate workflow and resource management and guarantee optimality or near optimality of cost functions for individual cases; (ii) how to guarantee consistency after a change in a workflow; and (iii) how to create a dynamic workflow. Static workflows can be described in WFDL.

Strong coordination models, whereby the process group P executes under the supervision of a coordinator process or processes. A coordinator process acts as an enactment engine and ensures a seamless transition from one process to another in the activity graph. 2. Weak coordination models, whereby there is no supervisory process.

There are similarities and some differences between workflows of traditional transaction-oriented systems and cloud workflows. The similarities are mostly at the modeling level, whereas the differences affect the mechanisms used to implement workflow management systems. Some of the more subtle differences between the two are.

The emphasis in a transactional model is placed on the contractual aspect of a transaction; in a workflow the enactment of a case is sometimes based on a "best-effort" model whereby the agents involved will do their best to attain the goal state but there is no guarantee of success.

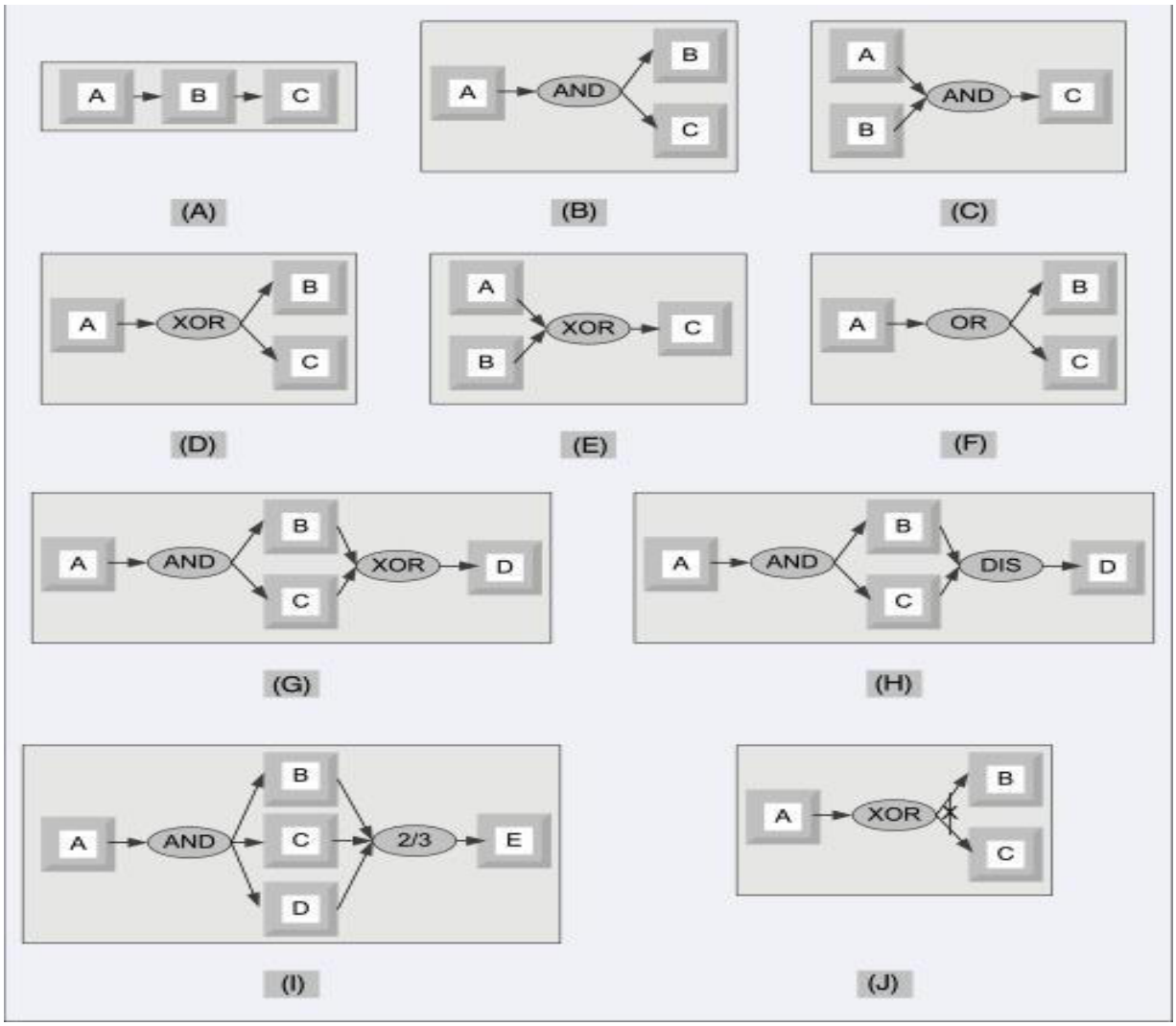
• A critical aspect of the transactional model in database applications is maintaining a consistent state of the database; however, a cloud is an open system, and thus its state is considerably more difficult to define.

• The database transactions are typically short-lived; the tasks of a cloud workflow could be longlasting.

• A database transaction consists of a set of well-defined actions that are unlikely to be altered during the execution of the transaction. However, the process description of a cloud workflow may change during the lifetime of a case.

• The individual tasks of a cloud workflow may not exhibit the traditional properties of database transactions. A task of a workflow could be either reversible or irreversible. Sometimes, paying a penalty for reversing an action is more profitable in the long run than continuing on a wrong path.

Resource allocation is a critical aspect of the workflow enactment on a cloud without an immediate correspondent for database transactions.



Basic workflow patterns. (a) Sequence. (b) AND split. (c) Synchronization. (d) XOR split. (e) XOR merge. (f)OR split. (g) Multiple merge. (h) Discriminator. (i) N out of M join. (j) Deferred choice.

Coordination based on a state machine model: The ZooKeeper

Cloud computing elasticity requires the ability to distribute computations and data across multiple systems. Coordination among these systems is one of the critical functions to be exercised in a distributed environment. The coordination model depends on the specific task, such as coordination of data storage, orchestration of multiple activities, blocking an activity until an event occurs, reaching consensus for the next action, or recovery after an error

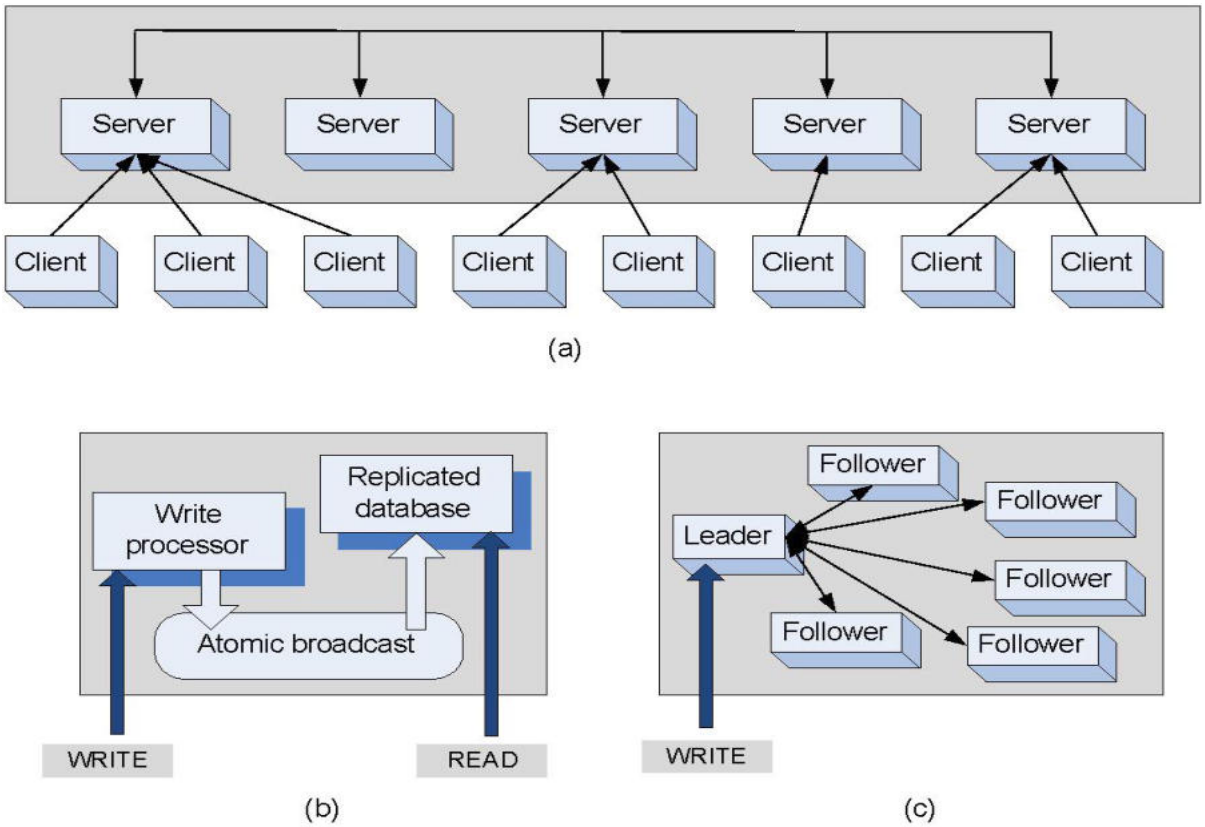
The entities to be coordinated could be processes running on a set of cloud servers or even running on multiple clouds. Servers running critical tasks are often replicated, so when one primary server fails, a backup automatically continues the execution. This is only possible if

the backup is in a hot standby mode – in other words, the standby server shares the same state at all times with the primary.

A solution for the proxy coordination problem is to consider a proxy as a deterministic finite state machine that performs the commands sent by clients in some sequence. The proxy has thus a definite state and, when a command is received, it transitions to another state. When P proxies are involved, all of them must be synchronized and must execute the same sequence of state machine command

ZooKeeper is a distributed coordination service based on this model. The high-throughput and low-latency service is used for coordination in large-scale distributed systems. The open-source software is written in Java and has bindings for Java and C.

The ZooKeeper software must first be downloaded and installed on several servers; then clients can connect to any one of these servers and access the coordination service. The service is available as long as the majority of servers in the pack are available. The organization of the service is shown in Figure 4.4. The servers in the pack communicate with one another and elect a leader. A database is replicated on each one of them and the consistency of the replicas is maintained. Figure 4.4(a) shows that the service provides a single system image. A client can connect to any server of the pack.



A client uses TCP to connect to a single server. Through the TCP connection a client sends requests and receives responses and watches events. A client synchronizes its clock with the server. If the server fails, the TCP connections of all clients connected to it time out and the clients detect the failure of the server and connect to other servers.

Figures 4.4(b) and (c) show that a read operation directed to any server in the pack returns the same result, whereas the processing of a write operation is more involved; the servers elect a leader, and any follower receiving a request from one of the clients connected to it forwards it to the leader. The leader uses atomic broadcast to reach consensus. When the leader fails, the servers elect a new leader.

The system is organized as a shared hierarchical namespace similar to the organization of a file system. A name is a sequence of path elements separated by a backslash. Every name in Zookeper's namespace is identified by a unique path.

In ZooKeeper the znodes, the equivalent of the inodes of a file system, can have data associated with them. Indeed, the system is designed to store state information.

A client can set a watch on a znode and receive a notification when the znode changes. This organization allows coordinated updates. The data retrieved by a client also contains a version number. Each update is stamped with a number that reflects the order of the transition.

The data stored in each node is read and written atomically. A read returns all the data stored in a znode, whereas a write replaces all the data in the znode. Unlike in a file system, Zookeeper data, the image of the state, is stored in the server memory. Updates are logged to disk for recoverability, and writes are serialized to disk before they are applied to the in-memory database that contains the entire tree. The ZooKeeper service guarantees:

1. Atomicity. A transaction either completes or fails.
2. Sequential consistency of updates. Updates are applied strictly in the order in which they are received.
3. Single system image for the clients. A client receives the same response regardless of the server it connects to.
4. Persistence of updates. Once applied, an update persists until it is overwritten by a client.
5. Reliability. The system is guaranteed to function correctly as long as the majority of servers function correctly.

The messaging layer is responsible for the election of a new leader when the current leader fails. The messaging protocol uses packets (sequences of bytes sent through a FIFO channel),

proposals (units of agreement), and messages (sequences of bytes atomically broadcast to all servers). A message is included in a proposal and it is agreed on before it is delivered

The application programming interface (API) to the ZooKeeper service is very simple and consists of seven operations:

• **create** – add a node at a given location on the tree.

• **delete** – delete a node.

• **get data** – read data from a node.

• **set data** – write data to a node.

• **get children** – retrieve a list of the children of the node. • synch – wait for the data to propagate.

This brief description shows that the ZooKeeper service supports the finite state machine model of coordination. In this case a znode stores the state. The ZooKeeper service can be used to implement higher-level operations such as group membership, synchronization, and so on. The system is used by Yahoo!'s Message Broker and by several other applications.

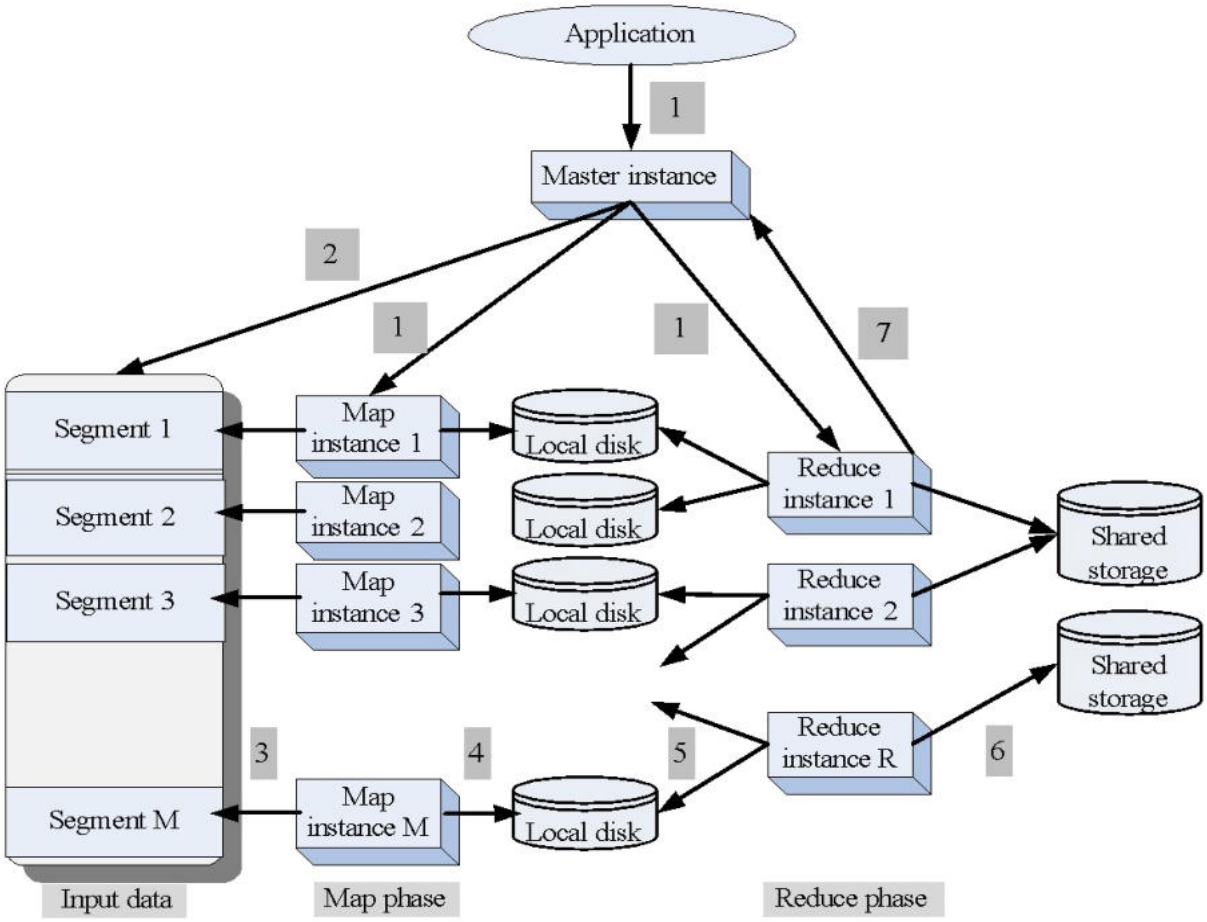
The MapReduce programming model

A main advantage of cloud computing is elasticity – the ability to use as many servers as necessary to optimally respond to the cost and the timing constraints of an application. In the case of transaction processing systems, typically a front-end system distributes the incoming transactions to a number of back-end systems and attempts to balance the load among them.

For data-intensive batch applications, partitioning the workload is not always trivial. Only in some cases can the data be partitioned into blocks of arbitrary size and processed in parallel by servers in the cloud. We distinguish two types of divisible workloads: • Modularly divisible. The workload partitioning is defined a priori. • Arbitrarily divisible. The workload can be partitioned into an arbitrarily large number of smaller workloads of equal or very close size.

MapReduce is based on a very simple idea for parallel processing of data-intensive applications supporting arbitrarily divisible load sharing. First, split the data into blocks, assign each block to an instance or process, and run these instances in parallel. Once all the instances have finished, the computations assigned to them start the second phase: Merge the partial results produced by individual instances. The so-called same program, multiple data (SPMD) paradigm, used since the early days of parallel computing, is based on the same idea but assumes that a master instance partitions the data and gathers the partial results.

MapReduce is a programming model inspired by the Map and the Reduce primitives of the LISP programming language. It was conceived for processing and generating large data sets on computing clusters. As a result of the computation, a set of input pairs is transformed into a set of output pairs.



The MapReduce philosophy. (1) An application starts a master instance and M worker instances for the Map phase and, later, R worker instances for the Reduce phase. (2) The master partitions the input data in M segments. (3) Each Map instance reads its input data segment and processes the data. (4) The results of the processing are stored on the local disks of the servers where the Map instances run. (5) When all Map instances have finished processing their data, the R Reduce instances read the results of the first phase and merge the partial results. (6) The final results are written by the Reduce instances to a shared storage server. (7) The master instance monitors the Reduce instances and, when all of them report task completion, the application is terminated

The following example shows the two user-defined functions for an application that counts the number of occurrences of each word in a set of documents.

map(String key, String value):

//key: document name; value: document contents for each word w in value: EmitIntermediate (w, "1");

reduce (String key, Iterator values):

// key: a word; values: a list of counts int result = 0; for each v in values: result += ParseInt (v); Emit (AsString (result));

Call M and R the number of Map and Reduce tasks, respectively, and N the number of systems used by the MapReduce. When a user program invokes the MapReduce function, the following sequence of actions take place.

1. The run-time library splits the input files into M splits of 16 to 64 MB each, identifies a number N of systems to run, and starts multiple copies of the program, one of the system being a master and the others workers. The master assigns to each idle system either a Map or a Reduce task
2. A worker being assigned a Map task reads the corresponding input split, parses pairs, and passes each pair to a user-defined Map function. The intermediate pairs produced by the Map function are buffered in memory before being written to a local disk and partitioned into R regions by the partitioning function.
3. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the Reduce workers. A Reduce worker uses remote procedure calls to read the buffered data from the local disks of the Map workers; after reading all the intermediate data, it sorts it by the intermediate keys.
4. When all Map and Reduce tasks have been completed, the master wakes up the user program.

ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES
RAJAMPET
(Autonomous)

Department of Artificial Intelligence & Machine Learning

Lecture Notes

Name of the Faculty: **Dr.T.Harikrishna** **Class:** **IVYear I Semester**

Branch and Section: **AIML** **Course code:** **20A57GT**

Name of the Course: Cloud Computing

Unit-III

Unit-3

Three classes of fundamental abstractions – interpreters, memory, and communications links – are necessary to describe the operation of a computing system. The physical realization of each one of these abstractions, such as processors that transform information, primary and secondary memory for storing information, and communication channels that allow different systems to communicate with one another, can vary in terms of bandwidth,1 latency,2 reliability, and other physical characteristics. Software systems such as operating systems are responsible for the management of the system resources –the physical implementations of the three abstractions**.**

The traditional solution for a data center is to install standard operating systems on individual systems and rely on conventional OS techniques to ensure resource sharing, application protection, and performance isolation. System administration, accounting, security, and resource management are very challenging for the providers of service in this setup; application development and performance Optimization are equally challenging for the users.

The alternative is resource virtualization, a technique analyzed in this chapter. Virtualization is a basic tenet of cloud computing – that simplifies some of the resource management tasks. For example,the state of a virtual machine (VM) running under a virtual machine monitor (VMM) can be saved and migrated to another server to balance the load. At the same time, virtualization allows users to operate in environments with which they are familiar rather than forcing them to work in idiosyncratic environments.

Resource sharing in a virtual machine environment requires not only ample hardware support and, in particular, powerful processors but also architectural support for multilevel control. Indeed, resources such as CPU cycles, memory,

secondary storage, and I/O and communication bandwidth are shared among several virtual machines; for each VM, resources must be shared among multiple instances of an application.

Traditional processor architectures were conceived for one level of control because they support two execution modes, the kernel and the user mode. In a virtualized environment all resources are under the control of a VMM and a second level of control is exercised by the guest operating system. Although two-level scheduling for sharing CPU cycles can be easily implemented, sharing of resources such as cache, memory, and I/O bandwidth is more intricate.

Virtualization

Virtualization simulates the interface to a physical object by any one of four means:

1.**Multiplexing.** Create multiple virtual objects from one instance of a physical object. For example, a processor is multiplexed among a number of processes or threads.

2. **Aggregation**. Create one virtual object from multiple physical objects. For example, a number of physical disks are aggregated into a RAID disk.

3. **Emulation.** Construct a virtual object from a different type of physical object. For example, a physical disk emulates a random access memory.

4.**Multiplexing and emulation**. Examples: Virtual memory with paging multiplexes real memory and disk, and a Virtual address emulates a real address; TCP emulates a reliable bit pipe and multiplexes a physical communication channel and a processor.

Virtualization abstracts the underlying resources and simplifies their use, isolates users from one another, and supports replication, which, in turn, increases the elasticity of the system. Virtualization is a critical aspect of cloud computing,

equally important to the providers and consumers of cloud services, and plays an important role in:

• System security because it allows isolation of services running on the same hardware.

• Performance and reliability because it allows applications to migrate from one platform to another.

• The development and management of services offered by a provider.

• Performance isolation.

User convenience is a necessary condition for the success of the utility computing paradigms. One of the multiple facets of user convenience is the ability to run remotely using the system software and libraries required by the application. User convenience is a major advantage of a VM architecture over a traditional operating system. For example, a user of the Amazon Web Services (AWS) could submit an Amazon Machine Image (AMI) containing the applications, libraries, data, and associated configuration settings. The user could choose the operating system for the application, then start, terminate, and monitor as many instances of the AMI as needed, using the Web Service APIs and the performance monitoring and management tools provided by the AWS.

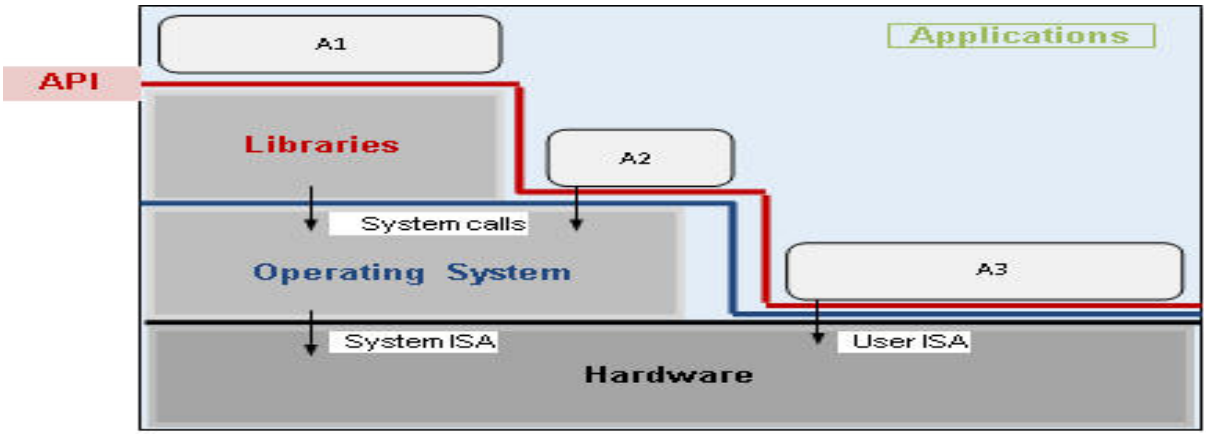
Layering and virtualization

A common approach to managing system complexity is to identify a set of layers with well-defined interfaces among them. The interfaces separate different levels of abstraction. Layering minimizes the interactions among the subsystems and simplifies the description of the subsystems. Each subsystem is abstracted through its interfaces with the other subsystems. Thus, we are able to design, implement, and modify the individual subsystems independently.

The instruction set architecture (ISA) defines a processor's set of instructions. For example, the Intel architecture is represented by the x86-32 and x86-64 instruction

sets for systems supporting 32-bit addressing and 64-bit addressing, respectively. The hardware supports two execution modes, a privileged, or kernel, mode and a user mode. The instruction set consists of two sets of instructions, privileged instructions that can only be executed in kernel mode and non-privileged instructions that can be executed in user mode. There are also sensitive instructions that can be executed in kernel and in user mode but that behave differently.

computer systems are fairly complex, and their operation is best understood when we consider a model similar to the one in Figure 5.1, which shows the interfaces among the software components and the hardware .The hardware consists of one or more multicore processors, a system interconnect (e.g., one or more buses), a memory translation unit, the main memory, and I/O devices, including one or more networking interfaces. Applications written mostly in high-level languages (HLL) often call library modules and are compiled into object code. Privileged operations, such as I/O requests, cannot be executed in user mode; instead, application and library modules issue system calls and the operating system determines whether the privileged operations required by the application do not violate system security or integrity and, if they don't, executes them on behalf of the user. The binaries resulting from the translation of HLL programs are targeted to a specific hardware architecture.



The first interface we discuss is the instruction set architecture (ISA) at the boundary of the hardware and the software. The next interface is the application binary interface (ABI), which allows the ensemble consisting of the application and the library modules to access the hardware. The ABI does not include privileged system instructions; instead it invokes system calls. Finally, the application program interface.

(API) defines the set of instructions the hardware was designed to execute and gives the application access to the ISA. It includes HLL library calls, which often invoke system calls. A process is the abstraction for the code of an application at execution time; a thread is a lightweight process. The ABI is the projection of the computer system seen by the process, and the API is the projection of the system from the perspective of the HLL program.

Clearly, the binaries created by a compiler for a specific ISA and a specific operating system are not portable. Such code cannot run on a computer with a different ISA or on computers with the same ISA but different operating systems. However, it is possible to compile an HLL program for a VM environment, as shown in Figure 5.2, where portable code is produced and distributed and then converted by binary translators to the ISA of the host system. A dynamic binary translation converts blocks of guest instructions from the portable code to the host instruction and leads to a significant performance improvement as such blocks are cached and reused.

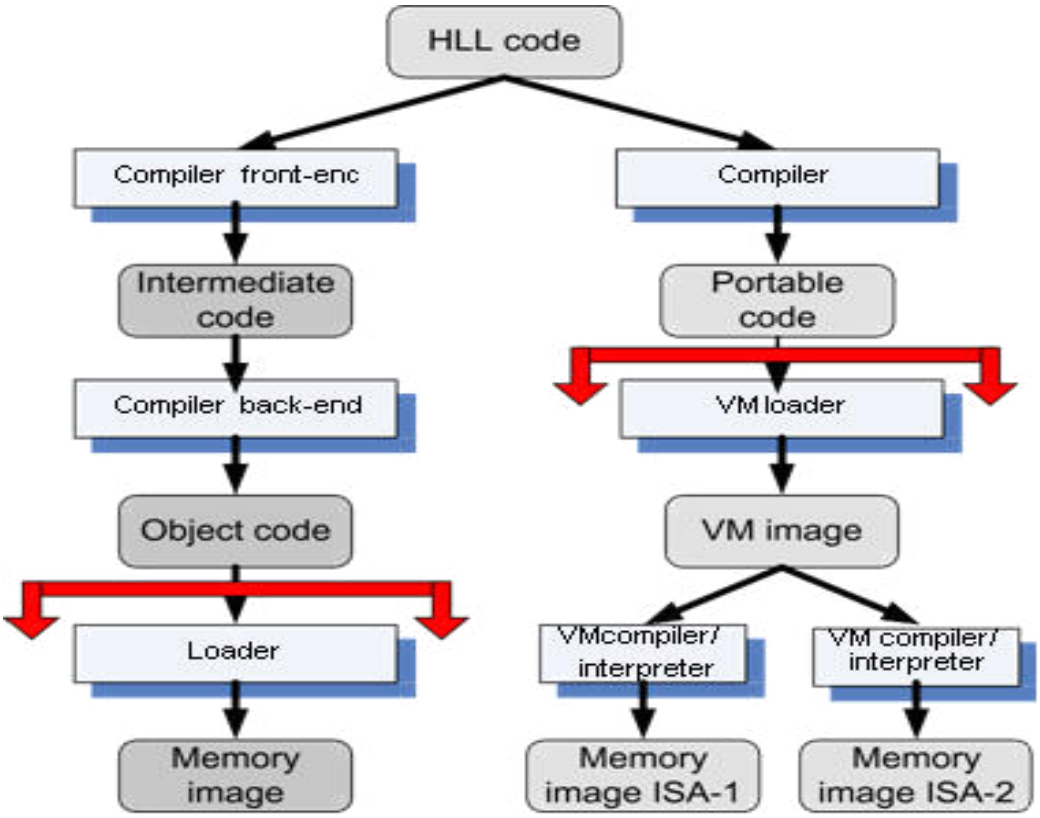


FIGURE 5.2

High-level language (HLL) code can be translated for a specific architecture and operating system. HLL code can also be compiled into portable code and then the portable code translated for systems with different ISAs. The code that is shared/distributed is the object code in the first case and the portable code in the second case

Virtual machine monitors

A virtual machine monitor (VMM), also called a hypervisor, is the software that securely partitions the resources of a computer system into one or more virtual machines. A guest operating system is an operating system that runs under the control of a VMM rather than directly on the hardware. The VMM runs in kernel mode, whereas a guest OS runs in user mode. Sometimes the hardware supports a third mode of execution for the guest OS.

VMMs allow several operating systems to run concurrently on a single hardware platform; at the same time, VMMs enforce isolation among these systems, thus enhancing security. A VMM controls how the guest operating system uses the

hardware resources. The events occurring in one VM do not affect any other VM running under the same VMM. At the same time, the VMM enables:

•Multiple services to share the same platform.

•The movement of a server from one platform to another, the so-called live migration.

•System modification while maintaining backward compatibility with the original system.

When a guest OS attempts to execute a privileged instruction, the VMM traps the operation and enforces the correctness and safety of the operation. The VMM guarantees the isolation of the individual VMs, and thus ensures security and encapsulation, a major concern in cloud computing. At the same time, the VMM monitors system performance and takes corrective action to avoid performance degradation; for example, the VMM may swap out a VM (copies all pages of that VM from real memory to disk and makes the real memory frames available for paging by other VMs) to avoid thrashing.

A VMM virtualizes the CPU and memory. For example, the VMM traps interrupts and dispatches them to the individual guest operating systems. If a guest OS disables interrupts, the VMM buffers such interrupts until the guest OS enables them. The VMM maintains a shadow page table for each guest OS and replicates any modification made by the guest OS in its own shadow page table. This shadow page table points to the actual page frame and is used by the hardware component called the memory management unit (MMU) for dynamic address translation.

Virtual machines

A virtual machine (VM) is an isolated environment that appears to be a whole computer but actually only has access to a portion of the computer resources. Each VM appears to be running on the bare hardware, giving the appearance of multiple instances of the same computer, though all are supported by a single physical

system. Virtual machines have been around since the early 1970s, when IBM released its VM/370 operating system.

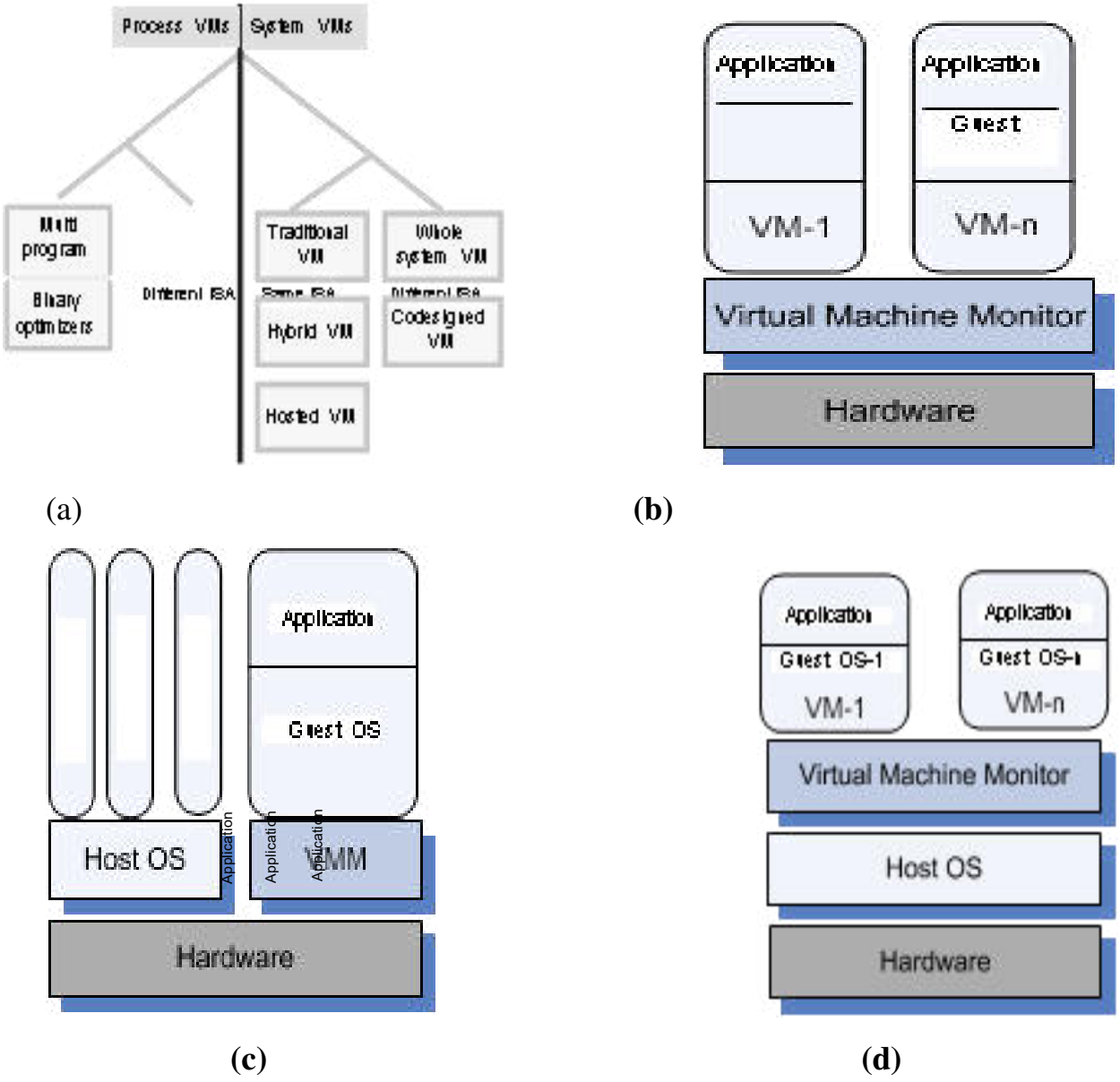


FIGURE 5.3

(a) A taxonomy of process and system VMs for the same and for different ISAs. Traditional, hybrid, and hosted are three classes of VM for systems with the same ISA. (b) Traditional VMs. The VMM supports multiple VMs and runs directly on the hardware. (c) A hybrid VM. The VMM shares the hardware with a host operating system and supports multiple virtual machines. (d) A hosted VM. The VMM runs under a host operating system.

We distinguish two types of VM: process and system VMs [see Figure 5.3(a)]. A process VM is a virtual platform created for an individual process and

destroyed once the process terminates. Virtually all operating systems provide a process VM for each one of the applications running, but the more interesting process VMs are those that support binaries compiled on a different instruction set. A system VM supports an operating system together with many user processes. When the VM runs under the control of a normal OS and provides a platform-independent host for a single application, we have an application virtual machine (e.g., Java Virtual Machine [JVM]).

Table 5.1 A nonexhaustive inventory of system virtual machines. The host ISA refers to the instruction set of the hardware; the guest ISA refers to the instruction set supported by the virtual machine. The VM could run under a host OS, directly on the hardware, or under a VMM. The guest OS is the operating system running under the control of a VM, which in turn may run under the control of the VMM.

Name	Host ISA	Guest ISA	Host OS	Guest OS	Company
Integrity VM	*x86*-64	*x* 86-64	HP-*Unix*	*Linux*, *Windows HP Unix*	HP
Power VM	Power	Power	No host OS	*Linux*, AIX	IBM IBM
z/VM	z-ISA	z-ISA	No host OS	*Linux* on z-ISA	LinuxWorks Microsoft
Lynx Secure	*x86*	*x86*	No host OS	*Linux*, *Windows*	Oracle Real Time Systems
Hyper-V Server	*x86*-64	*x86*-64	*Windows* No	*Windows*	SUN
Oracle VM	*x86*, *x86*-64	*x86*, *x86*-64	host OS No	*Linux*, *Windows*	VMware VMware
RTS Hypervisor	*x86*	*x86*	host OS	*Linux*, *Windows*	
					VMware VMware
SUN xVM	*x86*, SPARC	same as host	No host OS	*Linux*, *Windows*	
VMware EX Server	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	*Linux*, *Windows*, *Solaris*, *FreeBSD*	VMware
VMware Fusion	*x86*, *x86*-64	*x86*, *x86*-64	*Mac OS x86*	*Linux*, *Windows*, *Solaris*, *FreeBSD*	University of Washington
VMware Server	*x86*, *x86*-64	*x86*, *x86*-64	*Linux*, *Windows*	*Linux*, *Windows*, *Solaris*, *FreeBSD*	University of Cambridge
VMware Workstation	*x86*, *x86*-64	*x86*, *x86*-64	*Linux*, *Windows*	*Linux*, *Windows*, *Solaris*, *FreeBSD*	
VMware Player	*x86*, *x86*-64	*x86*, *x86*-64	*Linux*, *Windows*	*Linux*, *Windows*, *Solaris*, *FreeBSD*	
Denali	*x86*	*x86*	Denali	*ILVACO*, *NetBSD*	
Xen	*x86*, *x86*-64	*x86*, *x86*-64	*Linux Solaris*	*Linux*, *Solaris NetBSD*	

A literature search reveals the existence of some 60 different virtual machines, many created by the large software companies; Table 5.1 lists a subset of them. A system virtual machine provides a complete system; each VM can run its own OS, which in turn can run multiple applications. Systems such as Linux

Vserver], OpenVZ (Open VirtualiZation)], FreeBSD Jails [124], and Solaris Zones [296], based on Linux, FreeBSD, and Solaris, respectively, implement operating system-level virtualization technologies.

Operating system-level virtualization allows a physical server to run multiple isolated operating system instances, subject to several constraints; the instances are known as containers, virtual private servers (VPSs), or virtual environments (VEs). For example, OpenVZ requires both the host and the guest OS to be Linux distributions. These systems claim performance advantages over the systems based on a VMM such as Xen or VMware; according to [274], there is only a 1% to 3% performance penalty for OpenVZ compared to a stand-alone Linux server. OpenVZ is licensed under the GPL version 2.

Recall that a VMM allows several virtual machines to share a system. Several organizations of the software stack are possible:

•**Traditional**. VM also called a "bare metal" VMM. A thin software layer that runs directly on the host machine hardware; its main advantage is performance [see Figure 5.3(b)]. Examples: VMWare ESX, ESXi Servers, Xen, OS370, and Denali.

•**Hybrid**. The VMM shares the hardware with the existing OS [see Figure 5.3(c)]. Example: VMWare Workstation.

•**Hosted**. The VM runs on top of an existing OS [see Figure 5.3(d)]. The main advantage of this approach is that the VM is easier to build and install. Another advantage of this solution is that the VMM could use several components of the host OS, such as the scheduler, the pager, and the I/O drivers, rather than providing its own. A price to pay for this simplicity is the increased overhead and associated performance penalty; indeed, the I/O operations, page faults, and scheduling requests from a guest OS are not handled directly by the VMM. Instead, they are passed to the host OS. Performance as well as the challenges

to support complete isolation of VMs make this solution less attractive for servers in a cloud computing environment. Example: User-mode Linux.

Performance and security isolation

Performance isolation is a critical condition for quality-of-service (QoS) guarantees in shared computing environments. Indeed, if the run-time behavior of an application is affected by other applications running concurrently and, thus, is competing for CPU cycles, cache, main memory, and disk and network access, it is rather difficult to predict the completion time. Moreover, it is equally difficult to optimize the application. Several operating systems, including Linux/RK , QLinux and SILK , support some performance isolation, but problems still exist because one has to account for all resources used and to distribute the overhead for different system activities, including context switching and paging, to individual users – a problem often described as QoS crosstalk .

Processor virtualization presents multiple copies of the same processor or core on multicore systems. The code is executed directly by the hardware, whereas processor emulation presents a model of another hardware system in which instructions are "emulated" in software more slowly than virtualization. An example is Microsoft's Virtual PC, which could run on chip sets other than the x 86 families. It was used on Mac hardware until Apple adopted Intel chips.

Traditional operating systems multiplex multiple processes or threads, whereas a virtualization sup- ported by a VMM multiplexes full operating systems. Obviously, there is a performance penalty because an OS is considerably more heavy weight than a process and the overhead of context switching is larger. A VMM executes directly on the hardware a subset of frequently used machine instructions generated.

by the application and emulates privileged instructions, including device I/O requests. The subset of the instructions executed directly by the hardware includes arithmetic instructions, memory access, and branching instructions.

Operating systems use process abstraction not only for resource sharing but also to support isolation. Unfortunately, this is not sufficient from a security perspective. Once a process is compromised, it is rather easy for an attacker to penetrate the entire system. On the other hand, the software running on a virtual machine has the constraints of its own dedicated hardware; it can only access virtual devices emulated by the software. This layer of software has the potential to provide a level of isolation nearly equivalent to the isolation presented by two different physical systems. Thus, the virtualization can be used to improve security in a cloud computing environment.

Full virtualization and paravirtualization

In 1974 Gerald J. Popek and Robert P. Goldberg gave a set of sufficient conditions for a computer architecture to support virtualization and allow a VMM to operate efficiently [293]:

•A program running under the VMM should exhibit a behavior essentially identical to that demon- strated when the program runs directly on an equivalent machine.

•The VMM should be in complete control of the virtualized resources.

•A statistically significant fraction of machine instructions must be executed without the intervention of the VMM.

Another way to identify architecture suitable for a virtual machine is to distinguish two classes of machine instructions: sensitive instructions, which require special precautions at execution time, and innocuous instructions, which are not sensitive. In turn, sensitive instructions can be:

•**Control sensitive**, which are instructions that attempt to change either the memory allocation or the privileged mode.

•**Mode sensitive**, which are instructions whose behavior is different in the privileged mode.

An equivalent formulation of the conditions for efficient virtualization can be based on this classifi- cation of machine instructions. A VMM for a third-generation (or later) computer can be constructed if the set of sensitive instructions is a subset of the privileged instructions of that machine. To handle non virtualizable instructions, one could resort to two strategies:

•**Binary translation**. The VMM monitors the execution of guest operating systems; non virtualizable instructions executed by a guest operating system are replaced with other instructions.

•**Paravirtualization.**The guest operating system is modified to use only instructions that can be virtualized.

There are two basic approaches to processor virtualization: full virtualization, in which each virtual machine runs on an exact copy of the actual hardware, and paravirtualization, in which each virtual machine runs on a slightly modified copy of the actual hardware (see Figure 5.4). The reasons that paravirtualization is often adopted are (i) some aspects of the hardware cannot be virtualized; (ii) to improve performance; and (iii) to present a simpler interface

Full virtualization requires a virtualizable architecture; the hardware is fully exposed to the guest OS, which runs unchanged, and this ensures that this direct execution mode is efficient. On the other hand, paravirtualization is done because some architectures such as x86 are not easily virtualizable. Paravirtualization demands that the guest OS be modified to run under the VMM; furthermore, the guest OS code must be ported for individual hardware platforms.

Systems such as VMware EX Server support full virtualization on x86 architecture. The virtualization of the memory management unit (MMU) and the fact that privileged instructions executed by a guest OS fail silently pose some challenges

Application performance under a virtual machine is critical; generally, virtualization adds some level of overhead that negatively affects the performance. In some cases an application running under a VM performs better than one running under a classical OS. This is the case of a policy called cache isolation. The cache is generally not partitioned equally among processes running under a classical OS, since one process may use the cache space better than the other

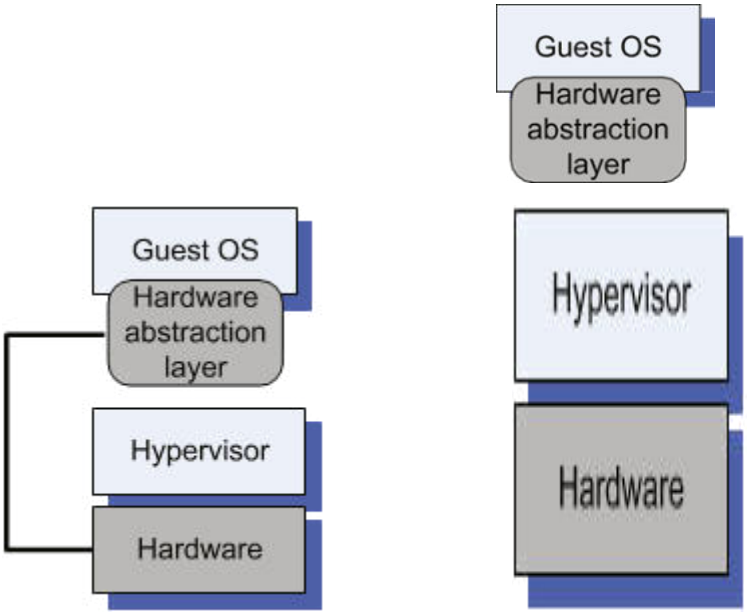


FIGURE 5.4

(a) Full virtualization requires the hardware abstraction layer of the guest OS to have some knowledge about the hardware. (b) Paravirtualization avoids this requirement and allows full compatibility at the application binary interface (ABI).

Hardware support for virtualization

In early 2000 it became obvious that hardware support for virtualization was necessary, and Intel and AMD started work on the first-generation virtualization extensions of the x86 3 architecture. In 2005 Intel released two Pentium 4 models supporting VT-x, and in 2006 AMD announced Pacifica and then several Athlon 64 models.

The challenges to virtualizing Intel architectures and then presents VT-x and VT-i virtualization architectures for x86 and Itanium architectures, respectively. Software solutions at that time addressed some of the challenges, but hardware solutions could improve not only performance but also security and, at the same time, simplify the software systems. We first examine the problems faced by virtualization of the x86 architecture.

•**Ring deprivileging**.This means that a VMM forces the guest software, the operating system, and the applications to run at a privilege level greater than 0. Recall that the x86 architecture provides four protection rings at levels 0–3. Two solutions are then possible: (a) The (0/1/3) mode, in which the VMM, the OS, and the application run at privilege levels 0, 1, and 3, respectively; or (b) the (0,3,3) mode, in which the VMM, a guest OS, and applications run at privilege levels 0, 3, and 3, respectively. The first mode is not feasible for x86 processors in 64-bit mode, as we shall see shortly.

•**Ring aliasing**. Problems created when a guest OS is forced to run at a privilege level other than that it was originally designed for. For example, when the CR register4 is PUSHed, the current privilege level is also stored on the stack.

•**Address space compression**. A VMM uses parts of the guest address space to store several system data structures, such as the interrupt-descriptor table and the global-descriptor table. Such data structures must be protected, but the guest software must have access to them.

•**Non faulting access to privileged state**. Several instructions, LGDT, SIDT, SLDT, and LTR that load the registers GDTR, IDTR, LDTR, and TR, can only be executed by software running at privilege level 0, because these instructions point to data structures that control the CPU operation Nevertheless, instructions that store from these registers fail silently when executed at a privilege level other than 0. This implies that a guest OS executing one of these instructions does not realize that the instruction has failed.

•**Guest system calls**. Two instructions, SYSENTER and SYSEXIT, support low-latency system calls. The first causes a transition to privilege level 0, whereas the second causes a transition from privilege level 0 and fails if executed at a level higher than 0. The VMM must then emulate every guest execution of either of these instructions, which has a negative impact on performance.

•**Interrupt virtualization**. In response to a physical interrupt, the VMM generates a "virtual interrupt" and delivers it later to the target guest OS. But every OS has the ability to mask interrupts5; thus the vir- tual interrupt could only be delivered to the guest OS when the interrupt is not masked. Keeping track of all guest OS attempts to mask interrupts greatly complicates the VMM and increases the overhead.

•**Access to hidden state**. Elements of the system state (e.g., descriptor caches for segment registers) are hidden; there is no mechanism for saving and restoring the hidden components when there is a context switch from one VM to another.

•**Ring compression**. Paging and segmentation are the two mechanisms to protect VMM code from being overwritten by a guest OS and applications. Systems running in 64-bit mode can only use paging, but paging does not distinguish among privilege levels 0, 1, and 2, so the guest OS must run at privilege level 3,

the so-called (0/3/3) mode. Privilege levels 1 and 2 cannot be used; thus the name ring compression.

•**Frequent access to privileged resources increases VMM overhead**. The task-priority register (TPR) is frequently used by a guest OS. The VMM must protect the access to this register and trap all attempts to access it. This can cause a significant performance degradation.

A major architectural enhancement provided by the VT-x is the support for two modes of operations and a new data structure called the virtual machine control structure (VMCS), including host-state and guest-state areas (see Figure 5.5):

•**VMX root**. Intended for VMM operations and very close to the x86 without VT-x.

•**VMX nonroot**. Intended to support a VM

When executing a VM entry operation, the processor state is loaded from the guest-state of the VM scheduled to run; then the control is transferred from the VMM to the VM. A VM exit saves the processor state in the guest-state area of the running VM; then it loads the processor state from the host-state area and finally transfers control to the VMM. Note that all VM exit operations use a common entry point to the VMM.

Each VM exit operation saves the reason for the exit and, eventually, some qualifications in VMCS. Some of this information is stored as bitmaps. For example, the exception bitmap specifies which one of 32 possible exceptions caused the exit. The I/O bitmap contains one entry for each port in a 16-bit I/O space.

The VMCS area is referenced with a physical address and its layout is not fixed by the architecture but can be optimized by a particular implementation. The VMCS includes control bits that facilitate the implementation of virtual interrupts.

Processors based on two new virtualization architectures, VT-d 6 and VT-c, have been developed.The first supports the I/O memory management unit (I/O MMU) virtualization and the second supports network virtualization.

Also known as PCI pass-through, I/O MMU virtualization gives VMs direct access to peripheral devices. VT-d supports:

•**DMA address remapping**, which is address translation for device DMA transfers.

•**Interrupt remapping**, which is isolation of device interrupts and VM routing.

•**I/O device assignment**, in which an administrator can assign the devices to a VM in any configuration.

•**Reliability features**, which report and record DMA and interrupt errors that may otherwise corrupt memory and impact VM isolation.

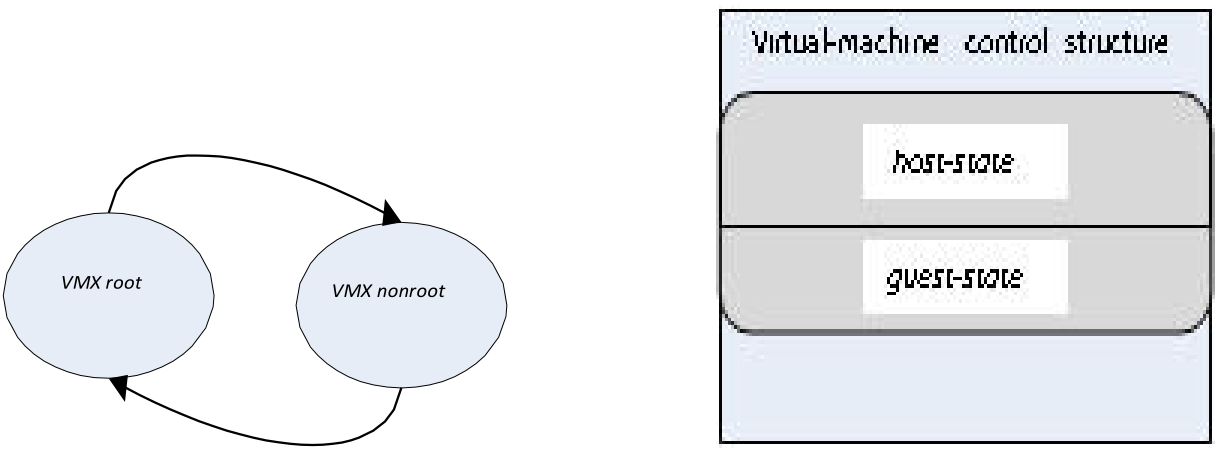


FIGURE 5.5

(a) The two modes of operation of VT-x, and the two operations to transit from one to another. (b) The VMCS includes host-state and guest-state areas that control the VM entry and VM exit transitions.

Case study: Xen, a VMM based on Para virtualization

Xen is a VMM or hypervisor developed by the Computing Laboratory at the University of Cambridge, United Kingdom, in 2003. Since 2010 Xen has been free software, developed by the community of users and licensed under the GNU General Public License (GPLv2). Several operating systems, including Linux, Minix, NetBSD, FreeBSD, NetWare, and OZONE, can operate as paravirtualized Xen guest operating systems running on x86, x86-64, Itanium, and ARM architectures

A VMM capable of scaling to about 100 VMs running standard applications and services without any modifications to the application binary interface (ABI). Fully aware that the x86 architecture does not support efficiently full virtualization, the designers of Xen opted for para virtualization.

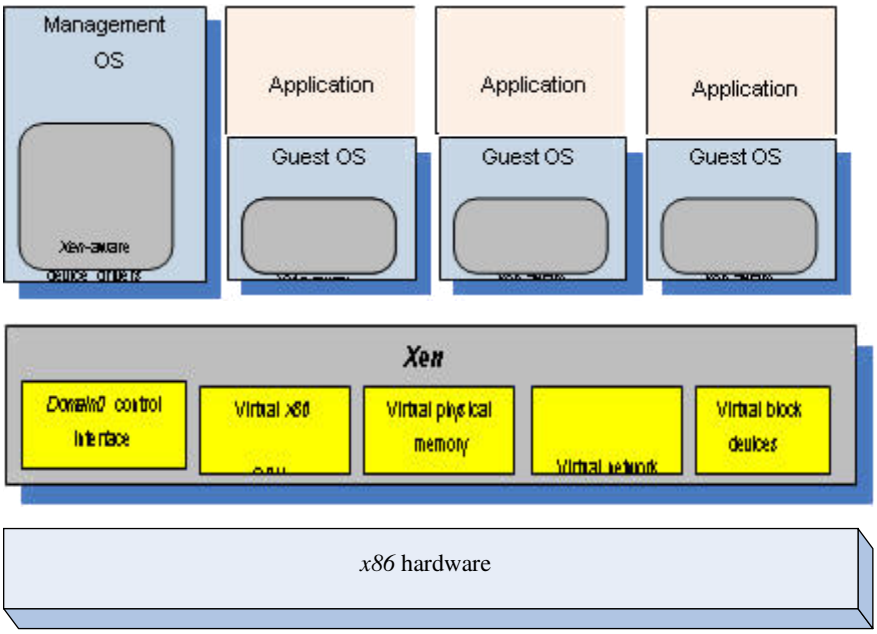


FIGURE 5.6

Xen for the x86 architecture. In the original Xen implementation [41] a guest OS could be XenoLinix, XenoBSD, or XenoXP. The management OS dedicated to the execution of Xen control functions and privileged instructions resides in Dom0; guest operating systems and applications reside in DomU.

guest OS and address spaces for applications running under this guest OS. Each domain runs on a virtual x86 CPU. Dom0 is dedicated to the execution of Xen control functions and privileged instructions, and DomU is a user domain (see Figure 5.6).

The most important aspects of the Xen para virtualization for virtual memory management, CPU multiplexing, and I/O device management are summarized in Table 5.2 [41]. Efficient management of the translation look-aside buffer (TLB), a cache for page table entries, requires either the ability to identify the OS and the address space of every entry or to allow software management of the TLB. Unfortunately, the x86 architecture does not support either the tagging of TLB entries or the software management of the TLB. As a result, address space switching, when the VMM activates a different OS, requires a complete TLB flush. This has a negative impact on performance.

The solution that was adopted was to load Xen in a 64 MB segment at the top of each address space and delegate the management of hardware page tables to the guest OS with minimal intervention from Xen. The 64 MB region occupied by Xen at the top of every address space is not accessible or not remappable by the guest OS. When a new address space is created, the guest OS allocates and initializes a page from its own memory, registers it with Xen, and relinquishes control of the write operations to the VMM. Thus, a guest OS could only map pages it owns. On the other hand, it has the ability to batch multiple page-update requests to improve performance. A similar strategy is used for segmentation.

Applications make system calls using the so-called hypercalls processed by Xen. Privileged instruc- tions issued by a guest OS are paravirtualized and must be validated by Xen. When a guest OS attempts to execute a privileged instruction directly, the instruction fails silently.

Memory is statically partitioned between domains to provide strong isolation. To adjust domain memory, XenoLinux implements a balloon driver, which passes pages between Xen and its own page allocator. For the sake of efficiency, page faults are handled directly by the guest OS.

Table 5.2 Paravirtualization strategies for virtual memory management, CPU multiplexing, and I/O devices for the original *x86 Xen* implementation.

Function	Strategy
Paging	A domain may be allocated discontinuous pages. A guest OS has direct access to page tables and handles page faults directly for efficiency. Page table updates are batched for performance and validated by *Xen* for safety.
Memory	Memory is statically partitioned between domains to provide strong isolation. XenoLinux implements a *balloon driver* to adjust domain memory.
Protection	A guest OS runs at a lower priority level, in ring 1, while *Xen* runs in ring 0.
Exceptions	A guest OS must register with *Xen* a description table with the addresses of exception handlers previously validated. Exception handlers other than the page fault handler are identical to *x*86 native exception handlers.
System calls	To increase efficiency, a guest OS must install a "fast" handler to allow system calls from an application to the guest OS and avoid indirection through *Xen*.
Interrupts	A lightweight event system replaces hardware interrupts. Synchronous system calls from a domain to *Xen* use *hypercalls*, and notifications are delivered using the asynchronous event system.
Multiplexing	A guest OS may run multiple applications.
Time	Each guest OS has a timer interface and is aware of "real" and "virtual" time.
Network and devices	Data is transferred using asynchronous I/O rings. A ring is a circular queue of descriptors I/O allocated by a domain and accessible within *Xen*.
Disk access	Only *Dom0* has direct access to IDE and SCSI disks. All other domains access persistent storage through the virtual block device (VBD) abstraction.

XenStore is a Dom0 process that supports a system-wide registry and naming service. It is imple- mented as a hierarchical key-value storage; a watch function of the process informs listeners of changes to the key in storage to which they have subscribed. XenStore communicates with guest VMs via shared memory using Dom0 privileges rather than grant tables.

The Toolstack is another Dom0 component responsible for creating, destroying, and managing the resources and privileges of VMs. To create a new VM a user provides a configuration file describing memory and CPU allocations as well as

device configuration.Then the Toolstack parses this file and writes this information in the **XenStore. Toolstack** takes advantage of Dom0 privileges to map guest memory, to load a kernel and virtual BIOS, and to set up initial communication channels with the XenStore and with the virtual console when a new VM is created Xen defines abstractions for networking and I/O devices. Split drivers have a front-end in the DomU and a back-end in Dom0; the two communicate via a ring in shared memory. Xen enforces access control for the shared memory and passes synchronization signals. Access control lists (ACLs) are stored in the form of grant tables, with permissions set by the owner of the memory.

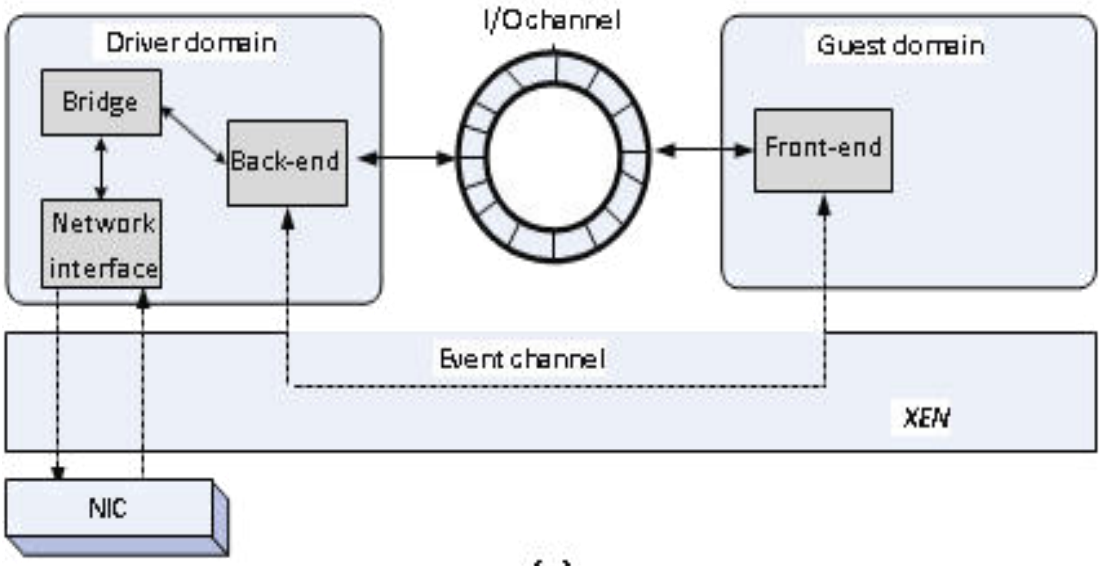
Data for I/O and network operations move vertically through the system very efficiently using a set of I/O rings (see Figure 5.7). A ring is a circular queue of descriptors allocated by a domain and accessible within Xen. Descriptors do not contain data; the data buffers are allocated off-band by the guest OS. Memory committed for I/O and network operations is supplied in a manner designed to avoid "cross-talk," and the I/O buffers holding the data are protected by preventing page faults of the corresponding page frame

Each domain has one or more virtual network interfaces (VIFs) that support the functionality of a network interface card. A VIF is attached to a virtual firewall-router (VFR). Two rings of buffer descriptors, one for packet sending and one for packet receiving, are supported. To transmit a packet, a guest OS enqueues a buffer descriptor to the send ring, then Xen copies the descriptor and checks safety and finally copies only the packet header, notthe payload, and executes the matching rules

The rules of the form (< pattern>, <action>) require the action to be executed if the pattern is matched by the information in the packet header. The rules can be added or removed by Dom0; they ensure the demultiplexing of packets based on the destination IP address and port and, at the same time, prevent spoofing of the

source IP address. Dom0 is the only one allowed to directly access the physical IDE (Integrated Drive Electronics) or SCSI (Small Computer System Interface) disks. All domains other than Dom0 access persistent storage through a virtual block device (VBD) abstraction created and managed under the control of Dom0.

Xen includes a device emulator, Qemu, to support unmodified commodity operating systems. Qemu emulates a DMA8 and can map any page of the memory in a DomU. Each VM has its own instance of Qemu that can run either as a Dom0 process or as a process of the VM.



(a)

An analysis of VM performance for I/O-bound applications under Xen is reported in [298]. Two Apache Web servers, each under a different VM, share the same server running Xen. The workload generator sends requests for files of fixed size ranging from 1 KB to 100 KB. When the file size increases from 1 KB to 10 KB and to 100 KB, the CPU utilization, throughput, data rate, and response time are, respectively: (97.5% 70.44% 44.4%), (1,900 1,104 112) requests/s, (2,018 11,048 11,208) KBps,and (1.52 2.36 2.08) msec. From the first group of results we see that for files 10 KB or larger the system is I/O bound; the second set of results shows that the throughput measured in requests/s decreases by less than 50% when the system becomes I/O bound, but the data rate increases by a factor of five over the same range. The variation of the response time is quite small; it increases about 10% when the file size increases by two orders of magnitude.

Optimization of network virtualization in Xen 2.0

A virtual machine monitor introduces a significant network communication overhead. For example, it is reported that the CPU utilization of a VMware Workstation 2.0 system running Linux 2.2.17 was 5 to 6 times higher than that of the native system (Linux 2.2.17) in saturating a 100 Mbps network [338].

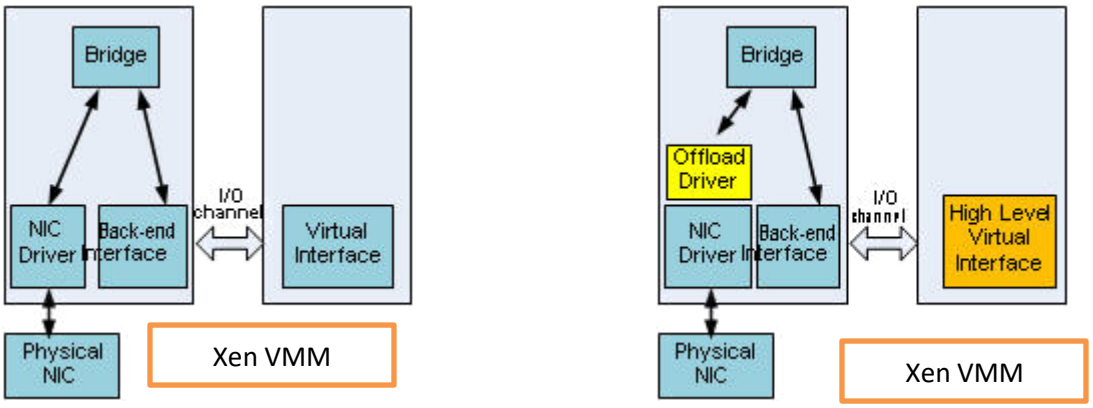
In other words, handling the same amount of traffic as the native system to saturate the network, the VMM executes a much larger number of instructions – 5 to 6 times larger.

Similar overheads are reported for other VMMs and, in particular, for Xen 2.0 . To under- stand the sources of the network overhead, we examine the basic network architecture of Xen [see Figure 5.8(a)]. Recall that privileged operations, including I/O, are executed by Dom0 on behalf of a guest operating system. In this context we shall refer to it as the driver domain called to execute networking operations on behalf of the guest domain. The driver domain uses the native Linux driver for the network interface controller, which in turn communicates with the physical NIC,

also called the network adapter. Recall from Section 5.8 that the guest domain communicates with the driver domain through an I/O channel; more precisely, the guest OS in the guest domain uses a virtual interface to send/receive data to/from the back-end interface in the driver domain.

Recall that a bridge in a LAN uses broadcast to identify the MAC address of a destination system. Once this address is identified, it is added to a table. When the next packet for the same destination arrives, the bridge uses the link layer protocol to send the packet to the proper MAC address rather than broadcast it. The bridge in the driver domain performs a multiplexing/demultiplexing function; packets received from the NIC have to be demultiplexed and sent to different VMs running under the VMM.

Table 5.3 shows the ultimate effect of this longer processing chain for the Xen VMM as well as the effect of optimizations [242]. The receiving and sending rates from a guest domain are roughly 30% and 20%, respectively, of the corresponding rates of a native Linux application. Packet multiplexing/



(a) (b)

FIGURE 5.8

Xen network architecture. (a) The original architecture. (b) The optimized architecture.

Table 5.3 shows the ultimate effect of this longer processing chain for the Xen VMM as well as the effect of optimizations [242]. The receiving and sending rates from a guest domain are roughly 30% and 20%, respectively, of the corresponding rates of a native Linux application. Packet multiplexing/ demultiplexing accounts for about 40% and 30% of the communication overhead for the incoming traffic and for the outgoing traffic, respectively.

Table 5.3 A comparison of send and receive data rates for a native *Linux* system, the *Xen* driver domain, an original *Xen* guest domain, and an optimized *Xen* guest domain.

System	Receive Data Rate (Mbps)	Send Data Rate (MBPS)
Linux	2,508	3,760
Xen driver	1,728	3,760
Xen guest	820	750
Optimized *Xen* guest	970	3,310

The Xen network optimization discussed in [242] covers optimization of (i) the virtual interface;(ii) the I/O channel; and (iii) the virtual memory. The effects of these optimizations are significant for the send data rate from the optimized Xen guest domain, an increase from 750 to 3, 310 Mbps, and rather modest for the receive data rate, 970 versus 820 Mbps.

Next we examine briefly each optimization area, starting with the **virtual interface**. There is a tradeoff between generality and flexibility on one hand and performance on the other hand. The original virtual network interface provides the guest domain with the abstraction of a simple low-level network interface supporting sending and receiving primitives. This design supports a wide range of physical devices attached to the driver domain but does not take advantage of the capabilities of some physical NICs such as checksum offload (e.g., TSO12) and scatter-gather DMA support.13 These features are supported by the high-level virtual interface of the optimized system

The next target of the optimization effort is the **communication between the guest domain and the driver domain.** Rather than copying a data buffer holding a packet, each packet is allocated in a new page and then the physical page containing the packet is remapped into the target domain. For example, when a packet is received, the physical page is remapped to the guest domain. The optimization is based on the observation that there is no need to remap the entire packet; for example, when sending a packet, the network bridge needs to know only the MAC header of the packet. As a result, the optimized implementation is based on an "out-of-band" channel used by the guest domain to provide the bridge with the packet MAC header. This strategy contributed to a better than four times increase in the send data rate compared with the non-optimized version

The third optimization covers **virtual memory**. Virtual memory in Xen 2.0 takes advantage of the superpage and global page-mapping hardware features available on Pentium and Pentium Pro proces- sors. A superpage increases the granularity of the dynamic address translation; a superpage entry covers 1, 024 pages of physical memory, and the address translation mechanism maps a set of contiguous pages to a set of contiguous physical pages. This helps reduce the number of TLB misses. Obviously, all pages of a superpage belong to the same guest OS. When new processes are created, the guest OS must allocate read-only pages for the page tables of the address spaces running under the guest OS, and that forces the system to use traditional page mapping rather than superpage mapping. The optimized version uses a special memory allocator to avoid this problem.

VBLADES: paravirtualization targeting an x86-64 Itanium processor

To understand the impact of computer architecture on the ability to efficiently virtualize a given archi- tecture, we discuss some of the findings of the vBlades project at HP-Laboratories [228]. The goal of the vBlades project was to create a VMM for the Itanium family of IA64 Intel processors,14 capable of supporting the

execution of multiple operating systems in isolated protection domains with security and privacy enforced by the hardware. The VMM was also expected to support optimal server utilization and allow comprehensive measurement and monitoring for detailed performance analysis.

We first review the features of the Itanium processor that are important for virtualization, starting with the observation that the hardware supports four privilege rings, PL0, PL1, PL2, and PL3. Privileged instructions can only be executed by the kernel running at level PL0, whereas applications run at level PL3 and can only execute nonprivileged instructions; PL2 and PL4 rings are generally not used. The VMM uses ring compression and runs itself at PL0 and PL1 while forcing a guest OS to run at PL2. A first problem, called privilege leaking, is that several nonprivileged instructions allow an application to determine the current privilege level (CPL); thus, a guest OS may not accept to boot or run or may itself attempt to make use of all four privilege rings.

Itanium was selected because of its multiple functional units and multithreading support. The Itanium processor has 30 functional units: six general-purpose ALUs, two integer units, one shift unit, four data cache units, six multimedia units, two parallel shift units, one parallel multiply, one population count, three branch units, two 82-bit floating-point multiply-accumulate units, and two SIMD floating-point multiply-accumulate units. A 128-bit instruction word contains three instructions; the fetch mechanism can read up to two instruction words per clock from the L1 cache into the pipeline. Each unit can execute a particular subset of the instruction set.

The Itanium processor supports isolation of the address spaces of different processes with eight privileged region registers. The Processor Abstraction Layer (PAL) firmware allows the caller to set the values in the region register. The VMM intercepts the privileged instruction issued by the guest OS to its PAL and

partitions the set of address spaces among the guest OSs to ensure isolation. Each guest is limited to 218 address spaces.

The hardware has an IVA register to maintain the address of the interruption vector table. The entries in this table control both the interrupt delivery and the interrupt state collection. Different types of interrupts activate different interrupt handlers pointed from this table, provided that the particular interrupt is not disabled. Each guest OS maintains its own version of this vector table and has its own IVA register. The hypervisor uses the guest OS IVA register to give control to the guest interrupt handler when an interrupt occurs.

First, let's discuss CPU virtualization. When a guest OS attempts to execute a privileged instruction, the VMM traps and emulates the instruction.There is a slight complication related to the fact that the Itanium does not have an instruction register (IR) and the VMM has to use state information to determine whether an instruction is privileged. Another complication is caused by the register stack engine (RSE), which operates concurrently with the processor and may attempt to access memory (load or store) and generate a page fault. Normally, the problem is solved by setting up a bit indicating that the fault is due to RSE and, at the same time, the RSE operations are disabled. The handling of this problem by the VMM is more intricate.

A number of privileged-sensitive instructions behave differently as a function of the privilege level. The VMM replaces each one of them with a privileged instruction during the dynamic transformation of the instruction stream. Among the instructions in this category are:

•**cover**, which saves stack information into a privileged register. The VMM replaces it with a break.b instruction.

•**thash and ttag**, which access data from privileged virtual memory control structures and have two registers as arguments. The VMM takes advantage of the

fact that an illegal read returns a zero and an illegal write to a register in the range 32 to 127 is trapped and translates these instructions as:
thash Rx=Ry –> tpa Rx=R(y+64) and ttag Rx=Ry –> tak Rx=R(y+64), where 0 ™ y ™ 64.

•**Access to performance** data from performance data registers is controlled by a bit in the processor status register with the PSR.sp instruction.

Memory virtualization is guided by the realization that a VMM should not be involved in most memory read and write operations to prevent a significant degradation of performance, but at the same time the VMM should exercise tight control and prevent a guest OS from acting maliciously. The vBlades VMM does not allow a guest OS to access the memory directly. It inserts an additional layer of indirection called metaphysical addressing between virtual and real addressing. A guest OS is placed in metaphysical addressing mode. If the address is virtual, the VMM first checks to see whether the guest OS is allowed to access that address and, if it is, it provides the regular address translation. If the address is physical the VMM is not involved. The hardware distinguishes between virtual and real addresses using bits in the processor status register.

A performance comparison of virtual machines

We have seen that a VMM such as Xen introduces additional overhead and negatively affects performance. The topic of this section is a quantitative analysis of the performance of VMs. We compare the performance of two virtualization techniques with a standard operating system: a plain-vanilla Linux referred to as “the base” system. The two VM systems are Xen, based on paravirtualization, and OpenVZ .

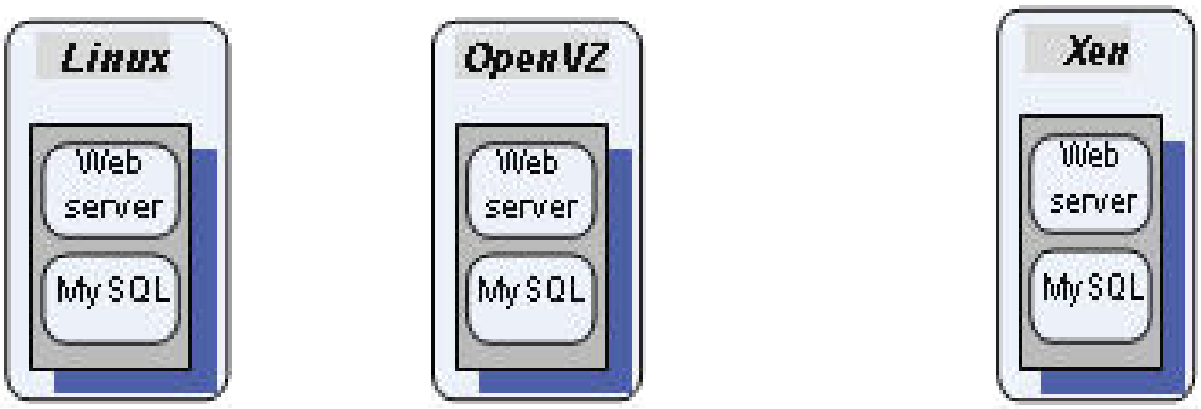
First we take a closer look at OpenVZ, a system based on OS-level virtualization. OpenVZ uses a single patched Linux kernel. The guest operating systems in different containers15 may be different distributions but must use the same Linux

kernel version that the host uses. The lack of flexibility of the approach for virtualization in OpenVZ is compensated by lower overhead.

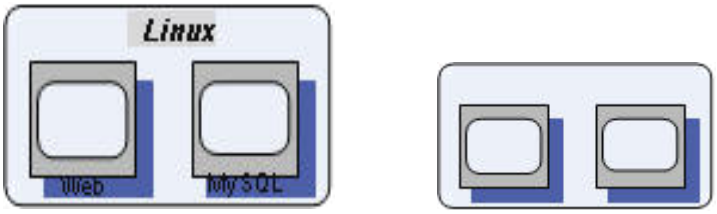
The memory allocation in OpenVZ is more flexible than in the case of paravirtualization; memory not used in one virtual environment can be used by others. The system uses a common file system. Each virtual environment is a directory of files isolated using chroot. To start a new virtual machine, one needs to copy the files from one directory to another, create a config file for the virtual machine, and launch the VM.

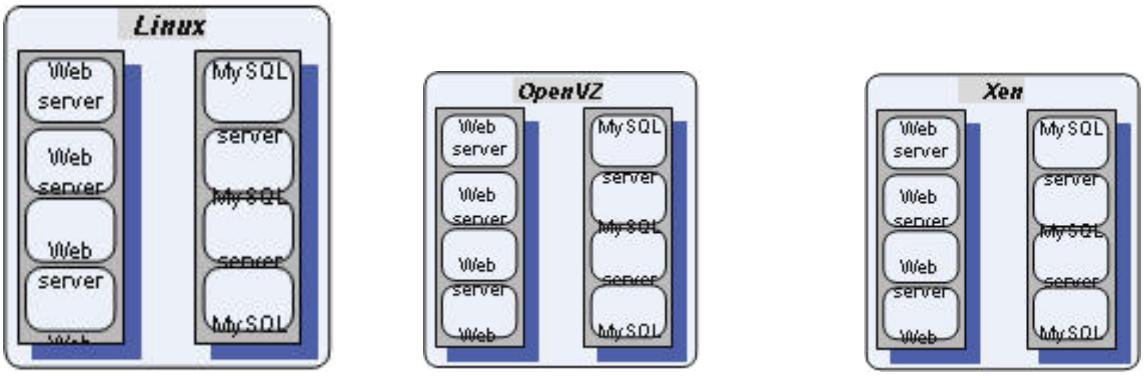
OpenVZ has a two-level scheduler: At the first level, the fair-share scheduler allocates CPU time slices to containers based on cpuunits values; the second level is a standard Linux scheduler that decides what process to run in that container. The I/O scheduler also has two levels; each container has an I/O priority, and the scheduler distributes the available I/O bandwidth according to the priorities.

There is ample experimental evidence that the load placed on system resources by a single application varies significantly in time. A time series displaying CPU consumption of a single application in time clearly illustrates this fact. As we all know, this phenomenon justifies the need for CPU multiplexing among threads/processes supported by an operating system. The concept of application and server consolidation is an extension of the idea of creating an aggregate load consisting of several applications and aggregating a set of servers to accommodate this load. Indeed, the peak resource requirements of individual applications are very unlikely to be synchronized, and the aggregate load tends to lead to a better average resource utilization.



(a)





(c)

reports the counters that allow the estimation of (i) the CPU time used by a binary; (ii) the number of L2-cache misses; and (iii) the number of instructions executed by a binary.

The experimental setups for three different experiments are shown in Figure 5.9 . In the first group of experiments the two tiers of the application, the Web and the DB, run on a single server for the Linux, the OpenVZ, and the Xen systems. When the workload increases from 500 to 800 threads, the throughput increases linearly

with the workload. The response time increases only slightly for the base system and for the OpenVZ system, whereas it increases 600% for the Xen system. For 800 threads the response time of the Xen system is four times longer than the time for OpenVZ. The CPU consumption grows linearly with the load in all three systems; the DB consumption represents only 1–4% of it.

For a given workload, the Web-tier CPU consumption for the OpenVZ system is close to that of the base system and is about half of that for the Xen system. The performance analysis tool shows that the OpenVZ execution has two times more L2-cache misses than the base system, whereas the Xen Dom0 has 2.5 times more and the Xen application domain has 9 times more. Recall that the base system and the OpenVZ run a Linux OS and the sources of cache misses can be compared directly, whereas Xen runs a modified Linux kernel. For the Xen-based system the procedure hypervisor _callback, invoked when an event occurs, and the procedure evtchn_do_upcall, invoked to process an event, are responsible for 32% and 44%, respectively, of the L2-cache misses. The first figure refers to the copying from user to system buffers and the second to copying from system buffers to the user space.

The second group of experiments uses two servers, one for the Web and the other for the DB application, for each one of the three systems. When the load increases from 500 to 800 threads the throughput increases linearly with the workload. The response time of the Xen system increases only 114%, compared with 600% reported for the first experiments. The CPU time of the base system, the OpenVZ system, the Xen Dom0, and the User Domain are similar for the Web application

For the DB application, the CPU time of the OpenVZ system is twice as long as that of the base system, whereas Dom0 and the User Domain require CPU times of 1.1 and 2.5 times longer than the base system.

The third group of experiments uses two servers, one for the Web and the other for the DB application, for each one of the three systems but runs four instances of the Web and the DB application on the two servers. The throughput increases linearly with the workload for the range used in the previous two experiments, from 500 to 800 threads. The response time remains relatively constant for OpenVZ and increases 5 times for Xen.

The main conclusion drawn from these experiments is that the virtualization overhead of Xen is con- siderably higher than that of OpenVZ and that this is due primarily to L2-cache misses. The performance degradation when the workload increases is also noticeable for Xen. Another important conclusion is that hosting multiple tiers of the same application on the same server is not an optimal solution.

The darker side of virtualization

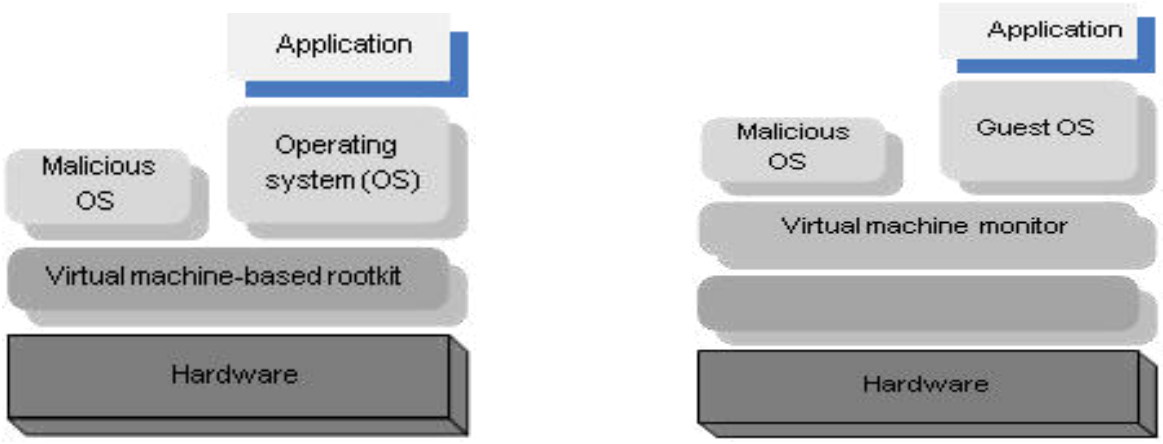
Can virtualization empower the creators of malware16 to carry out their mischievous activities with impunity and with minimal danger of being detected? How difficult is it to implement such a system?

It is well understood that in a layered structure a defense mechanism at some layer can be disabled by malware running a layer below it. Thus, the winner in the continuous struggle between the attackers and the defenders of a computing system is the one in control of the lowest layer of the software stack – the one that controls the hardware (see Figure 5.10).

Recall that a VMM allows a guest operating system to run on virtual hardware. The VMM offers to the guest operating systems a hardware abstraction and mediates its access to the physical hardware. We argued that a VMM is simpler and more compact than a traditional operating system; thus, it is more secure. But what if the VMM itself is forced to run above another software layer so that it is prevented from exercising direct control of the physical hardware

it is feasible to insert a "rogue VMM" between the physical hardware and an operating system. Such a rogue VMM is called a virtual machine-based rootkit (VMBR). The term rootkit refers to malware with privileged access to a system. The name comes from root, the most privileged account on a Unix system, and kit, a set of software components.

It is also feasible to insert the VMBR between the physical hardware and a "legitimate VMM." As a virtual machine running under a legitimate VMM sees virtual hardware, the guest OS will not notice any change of the environment; so the only trick is to present the legitimate VMM with a hardware abstraction, rather than allow it to run on the physical hardware



(a) **(b)**

FIGURE 5.10

The insertion of a VIRTUAL machine-based rootkit (VMBR) as the lowest layer of the software stack running on the physical hardware. (a) Below an operating system; (b) Below a legitimate virtual machine monitor. The VMBR enables a malicious OS to run surreptitiously and makes it invisible to the genuine or the guest OS and to the application.

The only way for a VMBR to take control of a system is to modify the boot sequence and to first load the malware and only then load the legitimate VMM or the operating system. This is only possible if the attacker has root privileges. Once the VMBR is loaded it must also store its image on the persistent storage.

The VMBR can enable a separate malicious OS to run surreptitiously and make this malicious OS invisible to the guest OS and to the application running under it. Under the protection of the VMBR, the malicious OS could (i) observe the data, the events, or the state of the target system;(ii) run services such as spam relays or distributed denial-of-service attacks; or (iii) interfere with the application

Software fault isolation

Software fault isolation (SFI) offers a technical solution for sandboxing binary code of questionable provenance that can affect security in cloud computing. Insecure and tampered VM images are one of the security threats because binary codes of questionable provenance for native plug-ins to a Web browser can pose a security threat when Web browsers are used to access cloud services.

ARM and 64-bit x86. ARM is a load/store architecture with 32-bit instruction and 16 general-purpose registers. It tends to avoid multicycle instructions, and it shares many RISC architecture features, but (a) it supports a “thumb” mode with 16-bit instruction extensions; (b) it has complex addressing modes and a complex barrel shifter; and (c) condition codes can be used to predicate most instructions. In the x86-64 architecture, general-purpose registers are extended to 64 bits.

This SFI implementation is based on the previous work of the same authors on Google Native Client (NC) and assumes an execution model in which a trusted run-time shares a process with an untrusted multithreaded plug-in. The rules for binary code generation of the untrusted plug-in are:(i) the code section is read-only and is statically linked; (ii) the code is divided into 32-byte bundles, and no instruction or pseudo-instruction crosses the bundle boundary; (iii) the disassembly

starting at the bundle boundary reaches all valid instructions; and (iv) all indirect flow-control instructions are replaced by pseudo-instructions that ensure address alignment to bundle boundaries.

Table 5.4 The features of the SFI for the native client on the *x86*-32, *x86*-64, and *ARM*. ILP stands for instruction-level parallelism.

Feature/Architecture	*x86*-32	*x86*-64	*ARM*
Addressable memory	1 GB	4 GB	1 GB
Virtual base address	Any	44 GB	0
Data model	ILP 32	ILP 32	ILP 32
Reserved registers	0 of 8	1 of 16	0 of 16
Data address mask	None	Implicit in result width	Explicit instruction
Control address mask	Explicit instruction	Explicit instruction	Explicit instruction
Bundle size (bytes)	32	32	16
Data in text segment	Forbidden	Forbidden	Allowed
Safe address registers	All	RSP, RBP	SP
Out-of-sandbox store	Trap	Wraps mod 4 GB	No effect
Out-of-sandbox jump	Trap	Wraps mod 4 GB	Wraps mod 1 GB

ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES
RAJAMPET
(Autonomous)

Department of Artificial Intelligence & Machine Learning

Lecture Notes

Name of the Faculty: **Dr.T.Harikrishna** **Class:** **IVYear I Semester**

Branch and Section: **AIML** **Course code:** **20A57GT**

Name of the Course: Cloud Computing

Unit-IV

Unit-4

Cloud Resource Management and Scheduling

Resource management is a core function of any man-made system. It affects the three basic criteria for the evaluation of a system: performance, functionality, and cost. An inefficient resource management has a direct negative effect on performance and cost and an indirect effect on the functionality of a system. Indeed, some functions provided by the system may become too expensive or may be avoided due to poor performance.

A cloud is a complex system with a very large number of shared resources subject to unpredictable requests and affected by external events it cannot control. Cloud resource management requires complex policies and decisions for multi-objective optimization. Cloud resource management is extremely challenging because of the complexity of the system, which makes it impossible to have accurate global state information, and because of the unpredictable interactions with the environment.

The strategies for resource management associated with the three cloud delivery models, IaaS, PaaS, and SaaS, differ from one another. In all cases the cloud service providers are faced with large, fluctuating loads that challenge the claim of cloud elasticity. In some cases, when a spike can be predicted, the resources can be provisioned in advance, e.g., for Web services subject to seasonal spikes. For an unplanned spike, the situation is slightly more complicated. Auto Scaling can be used for unplanned spike loads, provided that (a) there is a pool of resources that can be released or allocated on demand and (b) there is a monitoring system that allows a control loop to decide in real time to reallocate resources. Auto Scaling is supported by PaaS services such as Google App Engine.

Policies and mechanisms for resource management

A policy typically refers to the principal guiding decisions, whereas mechanisms represent the means to implement policies. Separation of policies from mechanisms is a guiding principle in computer science. Butler Lampson [208] and Per Brinch Hansen [154] offer solid arguments for this separation in the context of operating system design.

Cloud resource management policies can be loosely grouped into five classes:

1. Admission control.
2. Capacity allocation.
3. Load balancing.
4. Energy optimization.
5. Quality-of-service (QoS) guarantees.

The explicit goal of an **admission control policy** is to prevent the system from accepting workloads in violation of high-level system policies; for example, a system may not accept an additional workload that would prevent it from completing work already in progress or contracted. Limiting the workload requires some knowledge of the global state of the system. In a dynamic system such knowledge, when available, is at best obsolete. **Capacity allocation means** to allocate resources for individual instances; an instance is an activation of a service. Locating resources subject to multiple global optimization constraints requires a search of a very large search space when the state of individual systems changes rapidly.

Load balancing and energy optimization can be done locally, but global load-**balancing and energy optimization policies** encounter the same difficulties as the one we have already discussed. Load bal-ancing and energy optimization are correlated and affect the cost of providing the services.

The common meaning of the term load balancing is that of evenly distributing the load to a set of servers. For example, consider the case of four identical servers, A, B, C , and D, whose relative loads are 80%, 60%, 40%, and 20%, respectively, of their capacity. As a result of perfect load balancing, all servers would end with the same load 50% of each server's capacity. In cloud computing a critical goal is minimizing the cost of providing the service and, in particular, minimizing the energy consumption. This leads to a different meaning of the term load balancing; instead of having the load evenly distributed among all servers, we want to concentrate it and use the smallest number of servers while switching the others to standby mode, a state in which a server uses less energy.

Table 6.1 The normalized performance and energy consumption function of the processor speed. The performance decreases at a lower rate than does the energy when the clock rate decreases.

CPU Speed (GHz)	Normalized Energy *(%)*	Normalized Performance *(%)*
0.6	0.44	0.61
0.8	0.48	0.70
1.0	0.52	0.79
1.2	0.58	0.81
1.4	0.62	0.88
1.6	0.70	0.90
1.8	0.82	0.95
2.0	0.90	0.99
2.2	1.00	1.00

Virtually all optimal – or near-optimal – mechanisms to address the five classes of policies do not scale up and typically target a single aspect of resource management, e.g., admission control, but ignore energy conservation. Many require complex computations that cannot be done effectively in the time available to respond.

Allocation techniques in computer clouds must be based on a disciplined approach rather than ad hoc methods. The four basic mechanisms for the implementation of resource management policies are:

Control theory. Control theory uses the feedback to guarantee system stability and predict transient behavior, but can be used only to predict local rather than global behavior. Kalman filters have been used for unrealistically simplified models.

•**Machine learning**. A major advantage of machine learning techniques is that they do not need a performance model of the system.This technique could be applied to coordination of several autonomic system managers;

•**Utility-based**.Utility-based approaches require a performance model and a mechanism to correlate user-level performance with cost, as discussed in.

•**Market-oriented/economic mechanisms**. Such mechanisms do not require a model of the system, e.g., combinatorial auctions for bundles of resources discussed in.

Stability of a two-level resource allocation architecture

Two-level resource allocation architecture based on control theory concepts for the entire cloud. The automatic resource management is based on two levels of controllers, one for the service provider and one for the application, see Figure 6.2.

The main components of a control system are the inputs, the control system components, and the outputs. The inputs in such models are the offered workload and the policies for admission control, the capacity allocation, the load balancing, the energy optimization, and the QoS guarantees in the cloud. The system components are sensors used to estimate relevant measures of performance and controllers that implement various policies; the output is the resource allocations to the individual applications.

The controllers use the feedback provided by sensors to stabilize the system; stability is related to the change of the output. If the change is too large, the system may become unstable. In our context the system could experience thrashing, the amount of useful time dedicated to the execution of applications becomes

increasingly small and most of the system resources are occupied by management functions.

There are three main sources of instability in any control system:

1. **The delay** in getting the system reaction after a control action.

2. **The granularity** of the control, the fact that a small change enacted by the controllers leads to very large changes of the output.

3**. Oscillations**, which occur when the changes of the input are too large and the control is too weak, such that the changes of the input propagate directly to the output.

Two types of policies are used in autonomic systems: (i) threshold-based policies and (ii) sequential decision policies based on Markovian decision models.In the first case,upper and lower bounds on performance trigger adaptation through resource reallocation. Such policies are simple and intuitive but require setting per-application thresholds.

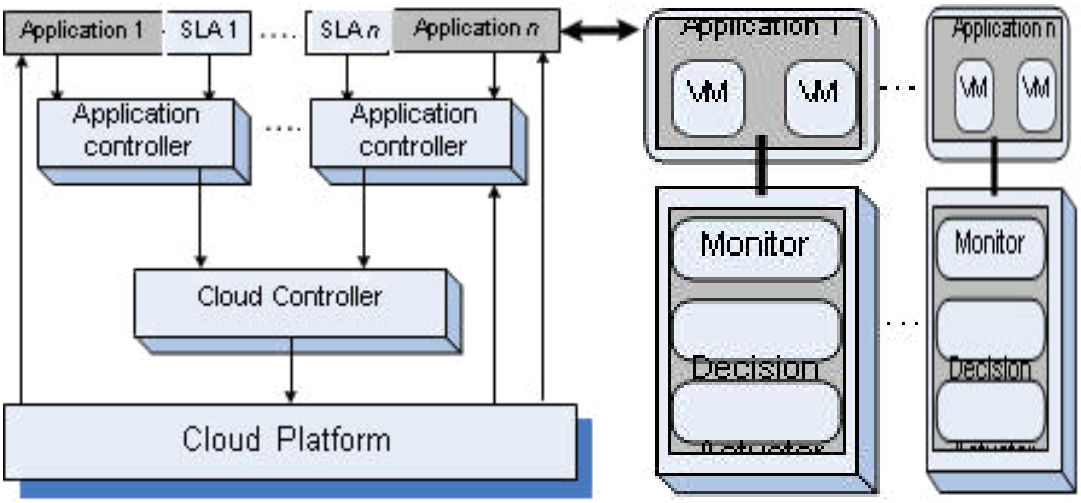


FIGURE 6.2

A two-level control architecture. Application controllers and cloud controllers work in concert.

Adjustments should be carried out only after the performance of the system has stabilized. The controller should measure the time for an application to stabilize and adapt to the manner in which the controlled system reacts.

If upper and lower thresholds are set, instability occurs when they are too close to one another if the variations of the workload are large enough and the time required to adapt does not allow the system to stabilize. The actions consist of allocation/deallocation of one or more virtual machines; sometimes allocation/deallocation of a single VM required by one of the thresholds may cause crossing of the other threshold and this may represent, another source of instability.

Feedback control based on dynamic thresholds

The elements involved in a control system are sensors, monitors, and actuators. The sensors measure the parameter(s) of interest, then transmit the measured values to a monitor, which determines whether the system behavior must be changed, and, if so, it requests that the actuators carry out the necessary actions. Often the parameter used for admission control policy is the current system load; when a threshold, e.g., 80%, is reached, the cloud stops accepting additional load

In practice, the implementation of such a policy is challenging or outright infeasible. First, due to the very large number of servers and to the fact that the load changes rapidly in time, the estimation of the current system load is likely to be inaccurate. Second, the ratio of average to maximal resource requirements of individual users specified in a service-level agreement is typically very high. Once an agreement is in place, user demands must be satisfied; user requests for additional resources within the SLA limits cannot be denied.

Thresholds. A threshold is the value of a parameter related to the state of a system that triggers a change in the system behavior. Thresholds are used in control theory to keep critical parameters of a system in a predefined range. The threshold could be static, defined once and for all, or it could be dynamic. A dynamic threshold

could be based on an average of measurements carried out over a time interval, a so-called integral control. The dynamic threshold could also be a function of the values of multiple parameters at a given time or a mix of the two.

To maintain the system parameters in a given range, a high and a low threshold are often defined. The two thresholds determine different actions; for example, a high threshold could force the system to limit its activities and a low threshold could encourage additional activities**. Control granularity** refers to the level of detail of the information used to control the system. **Fine control** means that very detailed information about the parameters controlling the system state is used, whereas **coarse control** means that the accuracy of these parameters is traded for the efficiency of implementation.

Proportional Thresholding.Application of these ideas to cloud computing, in particular to the IaaS delivery model, and a strategy for resource management called proportional thresholding are discussed in . The questions addressed are:

•Is it beneficial to have two types of controllers, (1) application controllers that determine whether additional resources are needed and (2) cloud controllers that arbitrate requests for resources and allocate the physical resources?

•Is it feasible to consider fine control? Is course control more adequate in a cloud computing environment.

Are dynamic thresholds based on time averages better than static ones?

• Is it better to have a high and a low threshold, or it is sufficient to define only a high threshold

The first two questions are related to one another. It seems more appropriate to have two controllers, one with knowledge of the application and one that’s aware of the state of the cloud. In this case a coarse control is more adequate for many reasons.

The essence of the proportional thresholding is captured by the following algorithm:

1. Compute the integral value of the high and the low thresholds as averages of the maximum and, respectively, the minimum of the processor utilization over the process history.

2. Request additional VMs when the average value of the CPU utilization over the current time slice exceeds the high threshold.

3. Release a VM when the average value of the CPU utilization over the current time slice falls below the low threshold.

Coordination of specialized autonomic performance managers

Virtually all modern processors support dynamic voltage scaling (DVS) as a mechanism for energy saving .Indeed, the energy dissipation scales quadratically with the supply voltage. The power management controls the CPU frequency and, thus, the rate of instruction execution. For some compute-intensive workloads the performance decreases linearly with the CPU clock frequency, whereas for others the effect of lower clock frequency is less noticeable or nonexistent. The clock frequency of individual blades/servers is controlled by a power manager, typically implemented in the firmware; it adjusts the clock frequency several times a second

The approach to coordinating power and performance management in is based on several ideas: • Use a join t utility function for power and performance. The joint performance-power utility function, Upp(R, P), is a function of the response time, R, and the power, P, and it can be of the form.

$$Upp(R, P) = U(R) - \times P \text{ or } Upp(R, P) = U(R)\ P$$

With U(R) the utility unction based on response time only and a parameter to weight the influence of the two factors, response time and power.

Identify a minimal set of parameters to be exchanged between the two managers.

• Set up a power cap for individual systems based on the utility-optimized power management policy. • Use a standard performance manager modified only to accept input from the power manager regarding the frequency determined according to the power management policy. The power manager consists of Tcl (Tool Command Language) and C programs to compute the per-server (per-blade) powercapsandsendthemviaIPMI5 to the firmware controlling the blade power. The power manager and the performance manager interact, but no negotiation between the two agents is involved.

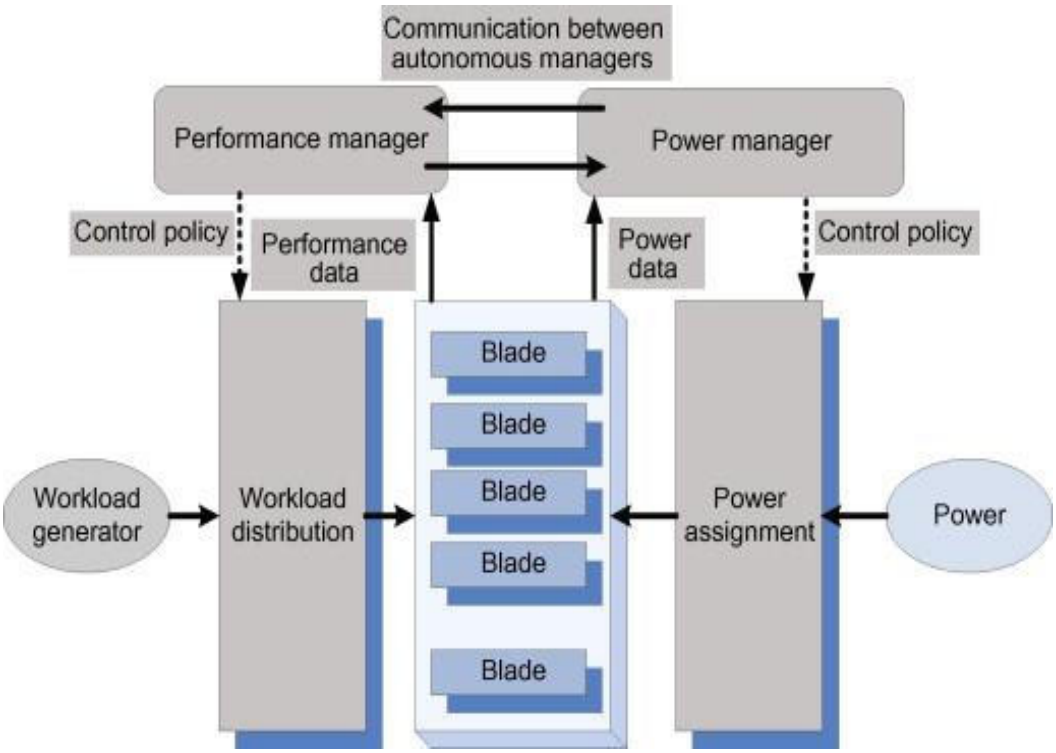
Use standard software systems. For example, use the WebSphere Extended Deployment (WXD), middleware that supports setting performance targets for individual Web applications and for the monitor response time, and periodically re compute the resource allocation parameters to meet the targets set. Use the Wide-Spectrum Stress Tool from the IBM Web Services Toolkit as a workload generator.

For practical reasons the utility function was expressed interms of nc, the number of clients, and pκ, the power cap, as in

$U^{l}(p\kappa,nc) = Upp(R(p\kappa,nc), P(p\kappa,nc))$.

The optimal powercap popt κ is a function of the workload intensity expressed by the number of clients,

$$nc, popt\ \kappa\ (nc) = argmax U^{l}(p\kappa,nc).$$



Storage systems

A cloud provides the vast amounts of storage and computing cycles demanded by many applications. The network-centric content model allows a user to access data stored on a cloud from any device connected to the Internet. Mobile devices with limited power reserves and local storage take advantage of cloud environments to store audio and video files. Clouds provide an ideal environment for multimedia content delivery.

Storage and processing on the cloud are intimately tied to one another indeed, sophisticated strategies to reduce the access time and to support real-time multimedia access are necessary to satisfy the requirements of content delivery. On the other hand, most cloud applications process very large amounts of data; effective data replication and storage management strategies are critical to the computations performed on the cloud.

A new concept, “big data,” reflects the fact that many applications use data sets so large that they cannot be stored and processed using local resources. The consensus is that “big data” growth can be viewed as a three-dimensional phenomenon; it implies an increased volume of data, requires an increased processing speed to process more data and produce more results, and at the same time it involves a diversity of data sources and data types.

Applications in many areas of science, including genomics, structural biology, high-energy physics, astronomy, meteorology, and the study of the environment, carry out complex analysis of data sets, often of the order of terabytes.1 In 2010, the four main detectors at the Large Hadron Collider (LHC) produced 13PB of data; the Sloan Digital Sky Survey (SDSS) collects about 200GB of data per night. As a result of this increasing appetite for data, file systems such as Btrfs,XFS,ZFS,exFAT,NTFS,HFS Plus, and ReFS support disk formats with theoretical volume sizes of several exa bytes.

The evolution of storage technology

The technological capacity to store information has grown over time at an accelerated pace.

• 1986: 2.6EB; equivalent to less than one 730MB CD-ROM of data per computer user.

• 1993: 15.8EB; equivalent to four CD-ROMs per user.

• 2000: 54.5EB; equivalent to 12 CD-ROMs per user.

• 2007: 295EB; equivalent to almost 61 CD-ROMs per user.

The density of Dynamic Random Access Memory(DRAM)increased from about 1 Gb/in2 in1990 to100 Gb/in2 in2003.The cost of DRAM tumbled from about $80/MB to less than $1/MB during the same period. In 2010 Samsung announced the first monolithic, 4gigabit, low-power, double-data-rate (LPDDR2) DRAM using a 30nm process. These rapid technological advancements have changed the

balance between initial investment in storage devices and system management costs. Now the cost of storage management is the dominant element of the total cost of a storage system. This effect favors the centralized storage strategy supported by a cloud; indeed, a centralized approach can automate some of the storage management functions, such as replication and backup, and thus reduce substantially the storage management cost.

The management of such a large collection of systems poses significant challenges and requires novel approaches to systems design. Case in point: Although the early distributed file systems used custom-designed reliable components, nowadays large-scale systems are built with off the-shelf components. The emphasis of the design philosophy has shifted from performance at any cost to reliability at the lowest possible cost. This shift is evident in the evolution of ideas, from the early distributed file systems of the 1980s,such as the Network File System(NFS) and the Andrew File System (AFS), to today's Google File System (GFS) and the Megastore.

Storage models, file systems, and databases

A storage model describes the layout of a data structure in physical storage; a data model captures the most important logical aspects of a datastructure in a database. The physical storage can be a local disk, a removable media, or storage accessible via a network.

Two abstract models of storage are commonly used: cell storage and journal storage. Cell storage assumes that the storage consists of cell of the same size and that each object fits exactly in one cell. This model reflects the physical organization of several storage media; the primary memory of a computer is organized as an array of memory cells, and a secondary storage device (e.g., a disk) is organized in sectors or blocks read and written as a unit. read/write

coherence and before-or-after atomicity are two highly desirable properties so fany storage model and in particular of cell storage.

Journal storage is a fairly elaborate organization for storing composite objects such as records consisting of multiple fields. Journal storage consists of a manager and cell storage, where the entire history of a variable is maintained, rather than just the current value. The user does not have direct access to the cell storage; instead the user can request the journal manager to (i) start a new action; (ii) read the value of a cell; (iii) write the value of a cell; (iv) commit an action; or (v) abort an action. The journal manager translates user requests to commands sent to the cell storage: (i) read a cell; (ii) write a cell; (iii) allocate a cell; or (iv) de allocate a cell.

An all-or-nothing action first records the action in a log in journal storage and then installs the change in the cell storage by over writing the previous version of a data item The log is always kept on nonvolatile storage (e.g., disk) and the considerably larger cell storage resides typically on nonvolatile memory, but can be held in memory for real-time access or using a write-through cache.

Many cloud applications must support online transaction processing and have to guarantee the correctness of the transactions. Transactions consist of multiple actions; for example, the transfer of funds from one account to another requires withdrawing funds from one account and crediting it to another. The system may fail during or after each one of the actions, and steps to ensure correctness must be taken. Correctness of a transaction means that the result should be guaranteed to be the same as though the actions were applied one after another, regardless of the order.

A file system consists of a collection of directories. Each directory provides information about a set of files. Today high-performance systems can choose

among three classes of file system: network file systems (NFSs), storage area networks (SANs), and parallel file systems (PFSs). The NFS is very popular and has been used for some time, but it does not scale well and has reliability problems; an NFS server could be a single point of failure.

SANs offer additional flexibility and allow cloud servers to deal with non-disruptive changes in the storage configuration. Moreover, the storage in a SAN can be pooled and then allocated based on the needs of the servers; pooling requires additional software and hardware support and represents another advantage of a centralized storage system. A SAN-based implementation of a file system can be expensive, since each node must have a Fibre Channel adapter to connect to the network.

Parallel file systems are scalable, are capable of distributing files across a large number of nodes, and provide a global naming space. In a parallel data system, several I/O nodes serve data to all computational nodes and include a metadata server that contains information about the data stored in the I/O nodes. The interconnection network of a parallel file system could be a SAN. Most cloud applications do not interact directly with file systems but rather through an application layer that manages a database. A database is a collection of logically related records. The software that controls the access to the database is called a database management system (DBMS). The main functions of a DBMS are to enforce data integrity, manage data access and concurrency control, and support recovery after a failure.

A DBMS supports a query language, a dedicated programming language used to develop database applications. Several database models, including the navigational model of the 1960s, the relational model of the 1970s, the object-oriented model of the 1980s, and the NoSQL model of the first decade of the 2000s, reflect the

limitations of the hardware available at the time and the requirements of the most popular applications of each period.

These requirements cannot be satisfied simultaneously by existing database models; for example, relational databases are easy to use for application development but do not scale well. As its name implies, the NoSQL model does not support SQL as a query language and may not guarantee the atomicity, consistency, isolation, durability (ACID) properties of traditional databases. NoSQL usually guarantees the eventual consistency for transactions limited to a single data item. The NoSQL model is useful when the structure of the data does not require a relational model and the amount of data is very large. Several types of NoSQL database have emerged in the last few years. Based on the way the NoSQL databases store data, we recognize several types, such as key-value stores, Big Table implementations, document store databases, and graph databases.

Distributed file systems: The precursors

The systems covered are the Network File System developed by Sun Microsystems in 1984, the Andrew File System developed at Carnegie Mellon University as part of the Andrew project, and the Sprite Network File System developed by John Osterhout's group at UC Berkeley as a component of the Unix-like distributed operating system called Sprite. Other systems developed at about the same time are Locus [365], Apollo [211], and the Remote File System (RFS) The main concerns in the design of these systems were scalability, performance, and security.

A majority of workstations were running under Unix; thus, many design decisions for the NFS were influenced by the design philosophy of the Unix File System (UFS). It is not surprising that the NFS designers aimed to: • Provide the same semantics as a local UFS to ensure compatibility with existing applications.

• Facilitate easy integration into existing UFS.

• Ensure that the system would be widely used and thus support clients running on different operating systems.

• Accept a modest performance degradation due to remote access over a network with a bandwidth of several Mbps.

Before we examine NFS in more detail, we have to analyze three important characteristics of the Unix File System that enabled the extension from local to remote file management:

• The layered design provides the necessary flexibility for the file system; layering allows separation of concerns and minimization of the interaction among the modules necessary to implement the system. The addition of the vnode layer allowed the Unix File System to treat local and remote file access uniformly.

• The hierarchical design supports scalability of the file system; indeed, it allows grouping of files into special files called directories and supports multiple levels of directories and collections of directories and files, the so-called file systems. The hierarchical file structure is reflected by the file-naming convention.

• The metadata supports a systematic rather than an ad hoc design philosophy of the file system. The so called inodes contain information about individual files and directories. The inodes are kept on persistent media, together with the data. Metadata includes the file owner, the access rights, the creation time or the time of the last modification of the file, the file size, and information about the structure of the file and the persistent storage device cells where data is stored. Metadata also supports device independence, a very important objective due to the very rapid pace of storage technology development.

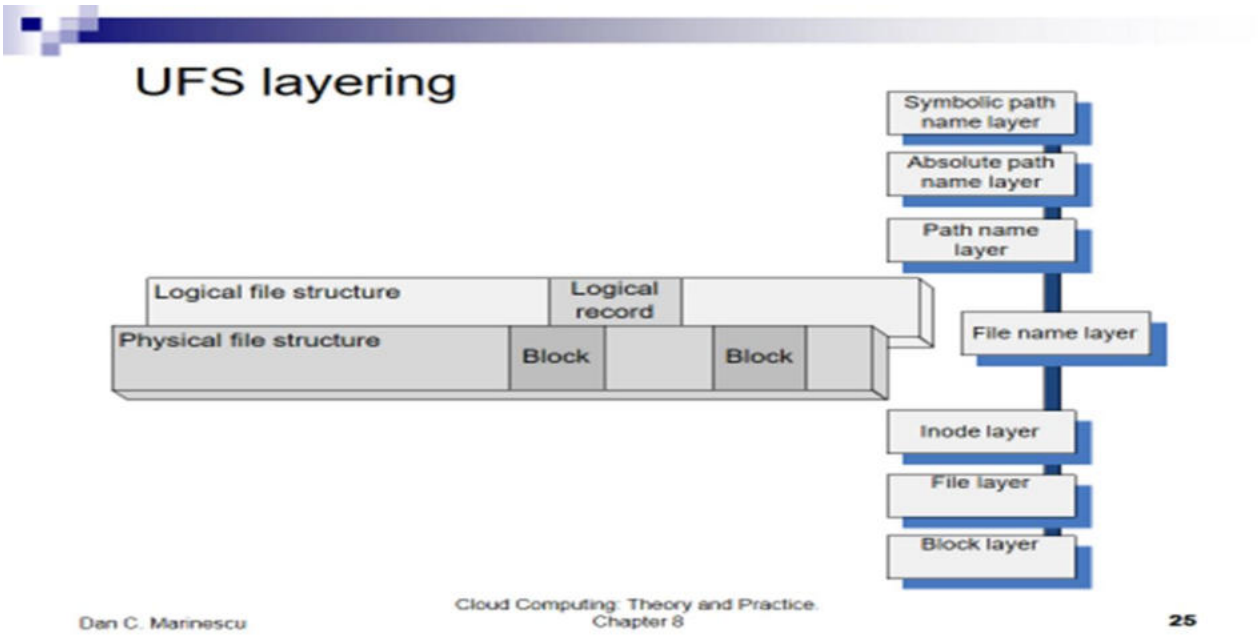
The logical organization of a file reflects the data model – the view of the data from the perspective of the application. The physical organization reflects the storage model and describes the manner in which the file is stored on a given storage medium. The layered design allows UFS to separate concerns for the

physical file structure from the logical one. Recall that a file is a linear array of cells stored on a persistent storage device. The file pointer identifies a cell used as a starting point for a read or write operation. This linear array is viewed by an application as a collection of logical records; the file is stored on a physical device as a set of physical records, or blocks, of a size dictated by the physical media.

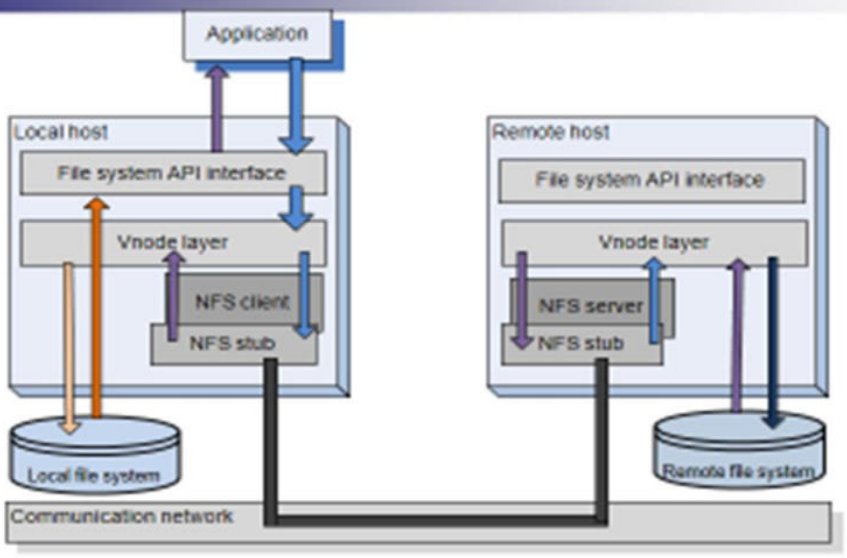
The lower three layers of the UFS hierarchy – the block, the file, and **the inode layer** – reflect the physical organization. **The block layer** allows the system to locate individual blocks on the physical device; the **file layer** reflects the organization of blocks into files; and the inode layer provides the metadata for the objects (files and directories). The upper three layers – the path name, the absolute path name, and the symbolic path name layer – reflect the logical organization. The filename layer mediates between the machine-oriented and the user-oriented views of the file system (see Figure 8.3).

Several control structures maintained by the kernel of the operating system support file handling by a running process. These structures are maintained in the user area of the process address space and can only be accessed in kernel mode. To access a file, a process must first establish a connection with the file system by opening the file. At that time a new entry is added to the file description table, and the meta-information is brought into another control structure, the open file table. A path specifies the location of a file or directory in a file system; a relative path specifies this location relative to the current/working directory of the process, whereas a full path, also called an absolute path, specifies the location of the file independently of the current directory, typically relative to the root directory. A local file is uniquely identified by a file descriptor (fd), generally an index in the open file table.

UFS layering



The Network File System is based on the client-server paradigm. The client runs on the local host while the server is at the site of the remote file system, and they interact by means of remote procedure calls (RPCs) (see Figure 8.4). The API interface of the local file system distinguishes file operations on a local file from the ones on a remote file and, in the latter case, invokes the RPC client. Figure 8.5 shows the API for aUnix File System, with the calls made by the RPC client in response to API calls issued by a user program for a remote file system as well as some of the actions carried out by the NFS server in response to an RPC call. NFS uses a vnode layer to distinguish between operations on local and remote files, as shown in Figure 8.4.

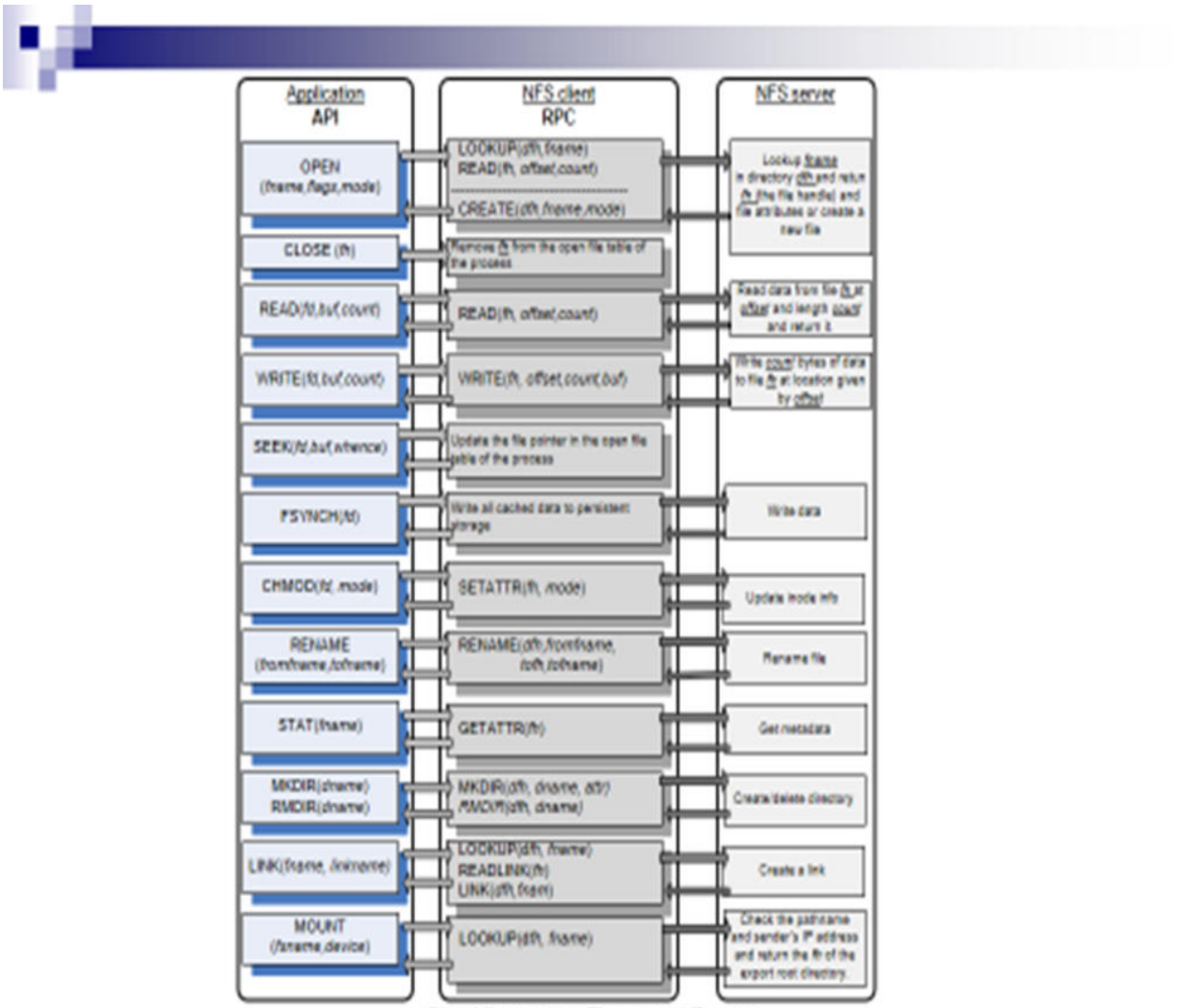


The NFS client-server interaction. The vnode layer implements file operation in a uniform manner, regardless of whether the file is local or remote.
An operation targeting a local file is directed to the local file system, while one for a remote file involves NFS; an NSF client packages the relevant information about the target and the NFS server passes it to the vnode layer on the remote host which, in turn, directs it to the remote file system.

A remote file is uniquely identified by a file handle (fh) rather than a file descriptor. The file handle is a 32-byte internal name, a combination of the file system identification, an inode number, and a generation number. The file handle allows the system to locate the remote file system and the file on that system; the generation number allows the system to reuse the inode numbers and ensures correct semantics when multiple clients operate on the same remote file.

Although many RPC calls, such as read,areidempotent,3 communication failures could sometimes lead to unexpected behavior. Indeed, if the network fails to deliver the response to a read RPC, then the call can be repeated without any side effects. By contrast, when the network fails to deliver the response to the rmdir

RPC, the second call returns an error code to the user if the call was successful the first time. If the server fails to execute the first call, the second call returns normally. Note also that there is no close RPC because this action only makes changes in the process open file structure and does not affect the remote file.



Cloud Computing: Theory and Practice.
Dan C. Marinescu Chapter 8 35

Andrew File System (AFS). AFS is a distributed file system developed in the late 1980s at Carnegie Mellon University (CMU) in collaboration with IBM [250]. The

designers of the system envisioned a very large number of workstations interconnected with a relatively small number of servers; it was anticipated that each individual at CMU would have an Andrew workstation, so the system would connect up to 10,000 workstations. The set of trusted servers in AFS forms a structure called Vice. The OS on a workstation, 4.2 BSD Unix, intercepts file system calls and forwards them to a user-level process called Venus, which caches files from Vice and stores modified copies of files back on the servers they came from. Reading and writing from/to a file are performed directly on the cached copy and bypass Venus; only when a file is opened or closed does Venus communicate with Vice. The emphasis of the AFS design was on performance, security, and simple management of the file system [170]. To ensure scalability and to reduce response time, the local disks of the workstations are used as persistent cache. The master copy of a file residing on one of the servers is updated only when the fileis modified. This strategy reduces the load placed on the servers and contributes to better system performance.

Another major objective of the AFS design was improved security. The communications between clients and servers are encrypted, and all file operations require secure network connections. When a user signs into a workstation, the password is used to obtain security tokens from an authentication server. These tokens are then used every time a file operation requires a secure network connection. The AFS uses access control lists (ACLs) to allow control sharing of the data. An ACL specifies the access rights of an individual user or group of users. A set of tools supports ACL management. Another facet of the effort to reduce user involvement in file management is location transparency. The files can be accessed from any location and can be moved automatically or at the request of system administrators without user involvement and/or inconvenience. The relatively small number of servers drastically reduces the efforts related to system

administration because operations, such as backups, affect only the servers, whereas workstations can be added, removed, or moved from one location to another without administrative intervention.

Sprite Network File System (SFS). SFS is a component of the Sprite network operating system.SFS supports non-write-through caching of files on the client as well as the server systems. Processes running on all workstations enjoy the same semantics for file access as they would if they were run on a single system.Thisispossibleduetoacacheconsistencymechanismthatflushesportions of the cache and disables caching for shared files opened for read/write operations. Caching not only hides the network latency, it also reduces server utilization and obviously improves performance by reducing response time. A file access request made by a client process could be satisfied at different levels. First, the request is directed to the local cache; if it's not satisfied there, it is passed to the local file system of the client .If it cannot be satisfied locally then the request is sent to the remote server. If the request cannot be satisfied by the remote server's cache,it is sent to the file system running on the server.

The design decisions for the Sprite system were influenced by the resources available at a time when a typical work station hada1–2MIPS processor and 4–14Mbytes of physical memory. The main-memory caches allowed diskless workstations to be integrated into the system and enabled the development of unique caching mechanisms and policies for both clients and servers. The results of a file-intensive benchmark report show that SFS was 30–35% faster than either NFS or AFS.

The file cache is organized as a collection of 4 KB blocks; a cache block has a virtual address consisting of a unique file identifier supplied by the server and a block number in the file. Virtual addressing allows the clients to create new blocks without the need to communicate with the server. File servers map virtual to

physical disk addresses. Note that the page size of the virtual memory in Sprite is also 4K. The size of the cache available to an SFS client or a server system changes dynamically as a function of the needs. This is possible because the Sprite operating system ensures optimal sharing of the physical memory between file caching by SFS and virtual memory management.

Thefilesystemandthevirtualmemorymanageseparatesetsofphysicalmemorypagesand maintain a time of last access for each block or page, respectively. Virtual memory uses a version of the clock algorithm [254] to implement a least recently used (LRU) page replacement algorithm, and the file system implements a strict LRU order, since it knows the time of each read and write operation. Whenever the file system or the virtual memory management experiences a file cache miss or a page fault, it compares the age of its oldest cache block or page, respectively, with the age of the oldest one of the other system; the oldest cache block or page is forced to release the real memory frame.

An important design decision related to the SFS was to delay write-backs; this means that a block is first written to cache, and the writing to the disk is delayed for a time of the order of tens of seconds. This strategy speeds up writing and avoids writing when the data is discarded before the time to write it to the disk. The obvious drawback of this policy is that data can be lost in case of a system failure. Write-through is the alternative to the delayed write-back; it guarantees reliability because the block is written to the disk as soon as it is available on the cache, but it increases the time for a write operation.

Most network files ystems guarantee that once a file is closed,the server will have the newest version on persistent storage.As far as concurrency is concerned,we distinguish sequential write sharing,when a file cannot be opened simultaneously for reading and writing by several clients, from concurrent write sharing, when multiple clients can modify the file at the same time. Sprite allows both modes of

concurrency and delegates the cache consistency to the servers. In case of concurrent write sharing, the client caching for the file is disabled; all reads and writes are carried out through the server.

Table 8.1 presents a comparison of caching, writing strategy, and consistency of NFS, AFS, Sprite, Locus, Apollo, and the Remote File System (RFS).

File system	Cache size and location	Writing policy	Consistency guarantees	Cache validation
NFS	Fixed, memory	On close or 30 sec. delay	Sequential	On open, with server consent
AFS	Fixed, disk	On close	Sequential	When modified server asks client
Sprite	Variable, memory	30 sec. delay	Sequential, concurrent	On open, with server consent
Locus	Fixed, memory	On close	Sequential, concurrent	On open, with server consent
Apollo	Variable, memory	Delayed or on unlock	Sequential	On open, with server consent
RFS	Fixed, memory	Write-through	Sequential, concurrent	On open, with server consent

General Parallel File System

Parallel I/O implies execution of multiple input/output operations concurrently. Support for parallel I/O is essential to the performance of many applications [236]. Therefore, once distributed file systems became ubiquitous, the natural next step in the evolution of the file system was to support parallel access. Parallel file systems allow multiple clients to read and write concurrently from the same file. Concurrency control is a critical issue for parallel file systems. Several semantics for handling the shared access are possible. For example, when the clients share the file pointer, successive reads issued by multiple clients advance the file pointer;

another semantic is to allow each client to have its own file pointer. Early supercomputers such as the Intel Paragon4 took advantage of parallel file systems to support applications based on the same program, multiple data (SPMD) paradigm.

The General Parallel File System (GPFS) [317] was developed at IBM in the early 2000s as a successor to the Tiger Shark multimedia file system [159]. GPFS is a parallel file system that emulates closely the behavior of a general-purpose POSIX system running on a single system.GPFS was designed for optimal performance of large clusters; it can support a file system of up to 4PB consisting of up to 4,096 disks of 1TB each.

Reliability is a major concern in a system with many physical components. To recover from system failures, GPFS records all metadata updates in a write-ahead log file. Write-ahead means that updates are written to persistent storage only after the log records have been written. For example, when a new file is created, a directory block must be updated and an inode for the file must be created. These records are transferred from cache to disk after the log records have been written. When the directory block is written and then the I/O node fails before writing the inode, then the system ends up in an inconsistent state and the log file allows the system to recreate the inode record.

The log files are maintained by each I/O node for each file system it mounts; thus, any I/O node is able to initiate recovery on behalf of a failed node. Disk parallelism is used to reduce access time. Multiple I/O read requests are issued in parallel and data is prefetched in a buffer pool.

Data striping allows concurrent access and improves performance but can have unpleasant side effects. Indeed, when a single disk fails, a large number of files are affected. To reduce the impact of such undesirable events, the system attempts to mask a single disk failure or the failure of the access path to a disk.

Consistency and performance, critical to any distributed file system, are difficult to balance. Support for concurrent access improves performance but faces serious challenges in maintaining consistency. In GPFS, consistency and synchronization are ensured by a distributed locking mechanism; a central lock manager grants lock tokens to local lock managers running in each I/O node. Lock tokens are also used by the cache management system.

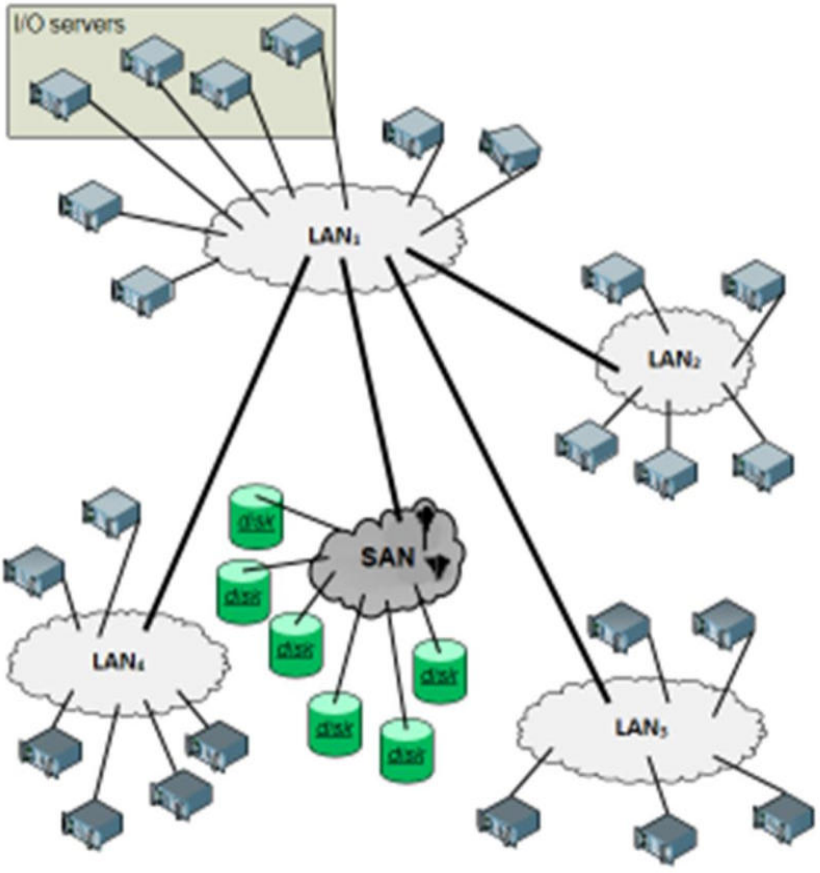
Lock granularity has important implications in the performance of a file system, and GPFS uses a variety of techniques for various types of data. Byte-range tokens are used for read and write operations to data files as follows: The first node attempting to write to a file acquires a token covering the entire file, $[0,\infty]$. This node is allowed to carry out all reads and writes to the file without any need for permission until a second node attempts to write to the same file. Then the range of the token given to the first node is restricted.

Byte-range token negotiations among nodes use the required range and the desired range for the offset and for the length of the current and future operations, respectively. Data shipping, an alternative to byte-range locking, allows fine-grained data sharing. In this mode the file blocks are controlled by the I/O nodes in a round-robin manner. A node forwards a read or write operation to the node controlling the target block, the only one allowed to access the file.

A token manager maintains the state of all tokens; it creates and distributes tokens, collects tokens once a file is closed, and downgrades or upgrades tokens when additional nodes request access to a file. Token management protocols attempt to reduce the load placed on the token manager; for example, when a node wants to revoke a token, it sends messages to all the other nodes holding the token and forwards the reply to the token manager.

GPFS uses disk maps to manage the disk space. The GPFS block size can be as large as 1MB, and a typical block size is 256KB. A block is divided into 32 sub

blocks to reduce disk fragmentation for small files; thus, the block map has 32bits to indicate whether a sub block is free or used. The system disk map is partitioned into n regions, and each disk map region is stored on a different I/O node. This strategy reduces conflicts and allows multiple nodes to allocate disk space at the same time. An allocation manager running on one of the I/O nodes is responsible for actions involving multiple disk map regions. For example, it updates free space statistics and helps with deallocation by sending periodic hints of the regions used by individual nodes.



Google File System

The Google File System (GFS) was developed in the late 1990s. It uses thousands of storage systems built from inexpensive commodity components to provide peta bytes of storage to a large user community with diverse needs. It is not surprising that a main concern of the GFS designers was to ensure the reliability of a system exposed to hardware failures, system software errors, application errors, and last but not least, human errors. The system was designed after a careful analysis of the file characteristics and of the access models. Some of the most important aspects of this analysis reflected in the GFS design are.

The system was designed after a careful analysis of the file characteristics and of the access models. Some of the most important aspects of this analysis reflected in the GFS design are:

• Scalability and reliability are critical features of the system; they must be considered from the beginning rather than at some stage of the design.

• The vast majority of files range in size from a few GB to hundreds of TB.

• The most common operation is to append to an existing file; random write operations to a file are extremely infrequent.

• Sequential read operations are the norm. • The users process the data in bulk and are less concerned with the response time. • The consistency mode lshould be relaxed to simplify the system implementation, but without placing an additional burden on the application developers.

Several design decisions were made as a result of this analysis:

1. Segment a file in large chunks.

2. Implement an atomic file append operation allowing multiple applications operating concurrently to append to the same file.

3. Build the cluster around a high-bandwidth rather than low-latency interconnection network Separate the flow of control from the data flow; schedule

the high-bandwidth data flow by pipelining the data transfer over TCP connections to reduce the response time. Exploit network topology by sending data to the closest node in the network.

4. Eliminate caching at the client site. Caching increases the overhead for maintaining consistency among cached copies at multiple client sites and it is not likely to improve performance.

5. Ensure consistency by channeling critical file operations through a master, a component of the cluster that controls the entire system.

6. Minimize the involvement of the master in file access operations to avoid hot-spot contention and to ensure scalability.

7. Support efficient check pointing and fast recovery mechanisms. 8. Support an efficient garbage-collection mechanism.

GFS files are collections of fixed-size segments called chunks; at the time of file creation each chunk is assigned a unique chunk handle. A chunk consists of 64 KB blocks and each block has a 32-bit checksum. Chunks are stored on Linux files systems and are replicated on multiple sites; a user may change the number of the replicas from the standard value of three to any desire dvalue.

The architecture of a GFS cluster is illustrated in Figure 8.7.Amaster controls a large number of chunk servers It maintains meta data such as file names, access control information, the location of all the replicas for every chunk of each file, and the state of individual chunk servers. Some of the metadata is stored in persistent storage.

The locations of the chunks are stored only in the control structure of the master's memory and are updated at system startup or when a new chunk server joins the cluster. This strategy allows the master to have up-to-date information about the location of the chunks. System reliability is a major concern and the operation log maintains a historical record of metadata changes,enabling the master to recover in

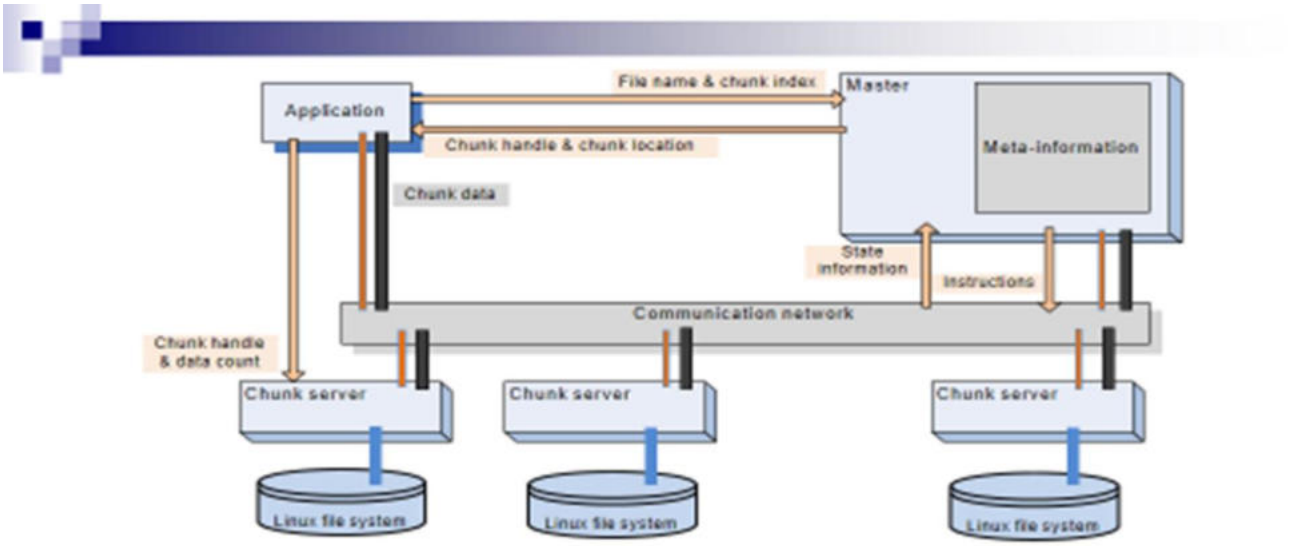
case of a failure. As a result, such changes are atomic and are not made visible to the clients until they have been recorded on multiple replicas on persistent storage. To recover from a failure, the master replays the operation log. To minimize the recovery time, the master periodically checkpoints its state and at recovery time replays only the log records after the last checkpoint.

When data for a write straddles the chunk boundary, two operations are carried out, one for each chunk. The steps for a write request illustrate a process that buffers data and decouples the control flow from the data flow for efficiency:

1. The client contacts the master, which assigns a lease to one of the chunk servers for a particular chunk if no lease for that chunk exists; then the master replies with the ID of the primary as well as secondary chunk servers holding replicas of the chunk. The client caches this information.
2. The client sends the data to all chunk servers holding replicas of the chunk; each one of the chunk servers stores the data in an internal LRU buffer and then sends an acknowledgment to the client.
3. The client sends a write request to the primary once it has received the acknowledgments from all chunk servers holding replicas of the chunk. The primary identifies mutations by consecutive sequence numbers.
4. The primary sends the write requests to all secondaries.
5. Each secondary applies the mutations in the order of the sequence numbers and then sends an acknowledgment to the primary. 6. Finally, after receiving the acknowledgments from all secondaries, the primary informs the client.

The system supports an efficient check pointing procedure based on copy-on-write to construct system snapshots. A lazy garbage collection strategy is used to reclaim the space after a file deletion. In the first step the filename is changed to a hidden name and this operation is time stamped. The master periodically scans the namespace and removes the metadata for the files with a hidden name older than a

few days; this mechanism gives a window of opportunity to a user who deleted files by mistake to recover the files with little effort.



- The architecture of a GFS cluster; the master maintains state information about all system components; it controls a number of chunk servers. A chunk server runs under Linux; it uses metadata provided by the master to communicate directly with the application. The data and the control paths are shown separately, data paths with thick lines and the control paths with thin lines. Arrows show the flow of control between the application, the master and the chunk servers.

Apache Hadoop

A wide range of data-intensive applications such as marketing analytics, image processing, machine learning, and Web crawling use Apache Hadoop,an open-source, Java-based software system. Hadoop supports distributed applications handling extremely large volumes of data. Many members of the community contributed to the development and optimization of Hadoop and several related Apache projects such as Hive and HBase.

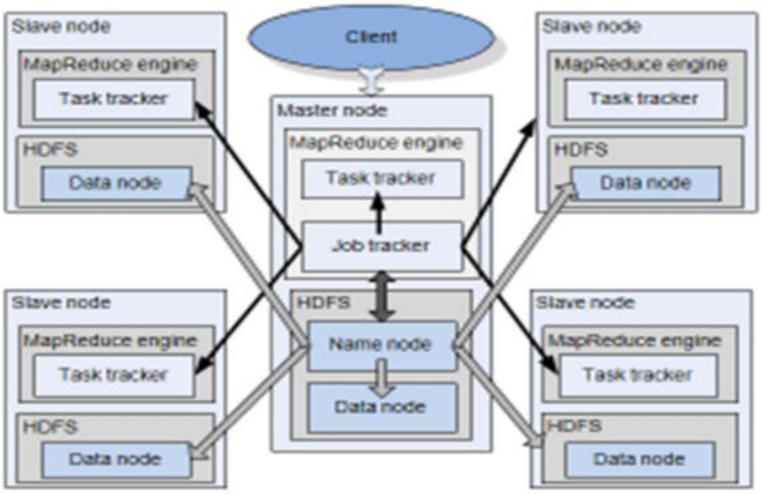
Hadoop is used by many organizations from industry, government, and research; the long list of Hadoop users includes major IT companies such as Apple, IBM, HP, Microsoft, Yahoo!, and Amazon; media companies such as The NewYork Time sand Fox; social networks, including Twitter,Facebook, and LinkedIn; and

government agencies, such as the U.S. Federal Reserve. In 2011, the Facebook Hadoop cluster had a capacity of 30PB.

A Hadoop system has two components, a MapReduce engine and a database (see Figure 8.8). The database could be the Hadoop File System (HDFS), Amazon S3, or Cloud Store, an implementation of the Google File System discussed in Section 8.5. HDFS is a distributed file system written in Java; it is portable, but it cannot be directly mounted on an existing operating system. HDFS is not fully POSIX compliant, but it is highly performant.

The Hadoop engine on the master of a multinode cluster consists of a job tracker and a task tracker, whereas the engine on a slave has only a task tracker. Thejob tracker receives a MapReduce job. from a client and dispatches the work to the task trackers running on the nodes of a cluster. To increase efficiency, the job tracker attempts to dispatch the tasks to available slaves closest to the placewhereitstoredthetaskdata.Thetasktrackersupervisestheexecutionoftheworkallocatedtothe node. Several scheduling algorithms have been implemented in Hadoop engines, including Facebook's fair scheduler and Yahoo!'s capacity scheduler; see Section 6.8 for a discussion of cloud scheduling algorithms.

HDFS replicates data on multiple nodes. The default tis three replicas a large data set is distributed over many nodes. The name node running on the master manages the data distribution and data replication and communicates with data nodes running on all cluster nodes; it shares with the job tracker information about data placement to minimize communication between the nodes on which data is located and the ones where it is needed. Although HDFS can be used for applications other than those based on the MapReduce model, its performance for such applications is not at par with the ones for which it was originally designed.



A Hadoop cluster using HDFS; the cluster includes a master and four slave nodes. Each node runs a MapReduce engine and a database engine. The job tracker of the master's engine communicates with task trackers on all the nodes and with the name node of HDFS. The name node of the HDFS shares information about the data placement with the job tracker to minimize communication between the nodes where data is located and the ones where it is needed.

Locks and Chubby:

A locking service Locks support the implementation of reliable storage for loosely coupled distributed systems; they enable controlled access to shared storage and ensure atomicity of read and write operations. Furthermore, critically important to the design of reliable distributed storage systems are distributed consensus problems, such as the election of a master from a group of data servers. A master has an important role in system management; for example, in GFS the master maintains state information about all system components.

Locks that can be held for only a very short time are called fine-grained, whereas coarse-grained locks are held for a longer time. Some operations require meta-information about a lock, such as the name of the lock, whether the lock is shared or held in exclusivity, and the generation number of the lock. This meta-information is sometimes aggregated into an opaque byte string called a sequencer.

The question of how to most effectively support a locking and consensus component of a large-scale distributed system demands several design decisions. A first design decision is whether **the locks should be mandatory or advisory**. Mandatory locks have the obvious advantage of enforcing access control; a traffic analogy is that a mandatory lock is like a drawbridge. Once it is up, all traffic is forced to stop. An advisory lock is like a stop sign those who obey the traffic laws will stop, but some might not. The disadvantages of mandatory locks are added overhead and less flexibility. Once a data item is locked, even a high-priority task related to maintenance or recovery cannot access the data unless it forces the applicationholdingthelocktoterminate.Thisisaverysignificantprobleminlarge-scalesystemswhere partial system failures are likely.

A second design decision is whether the system should be based on fine-grained or coarse-grained locking. Fine-grained locks allow more application threads to access shared data in any time interval, but they generate a larger work load for the lock server. Moreover, when the lock server fails for a period of time, a larger number of applications are affected. Advisory locks and coarse-grained locks seem to be a better choice for a system expected to scale to a very large number of nodes distributed in data centers that are interconnected via wide area networks.

A third design decision is how to support a systematic approach to locking. Two alternatives come to mind: (i) delegate to the clients the implementation of the consensus algorithm and provide a library of functions needed for this task, or (ii) create a locking service that implements a version of the asynchronous Paxos algorithm and provide a library to be linked with an application client to support service calls.Forcing application developers to invoke calls to a Paxos library is more cumbersome and more prone to errors than the service alternative. Of course, the lock service itself has to be scalable to support a potentially heavy load.

Another design consideration is flexibility, the ability of the system to support a variety of applications. A name service comes immediately to mind because many cloud applications read and write small files. The names of small files can be included in the namespace of the service to allow atomic file operations.The choice should also consider the performance a service can be optimized and clients can cache control information. Finally, we should consider the overhead and resources for reaching consensus. Again, the service seems to be more advantageous because it needs fewer replicas for high availability.

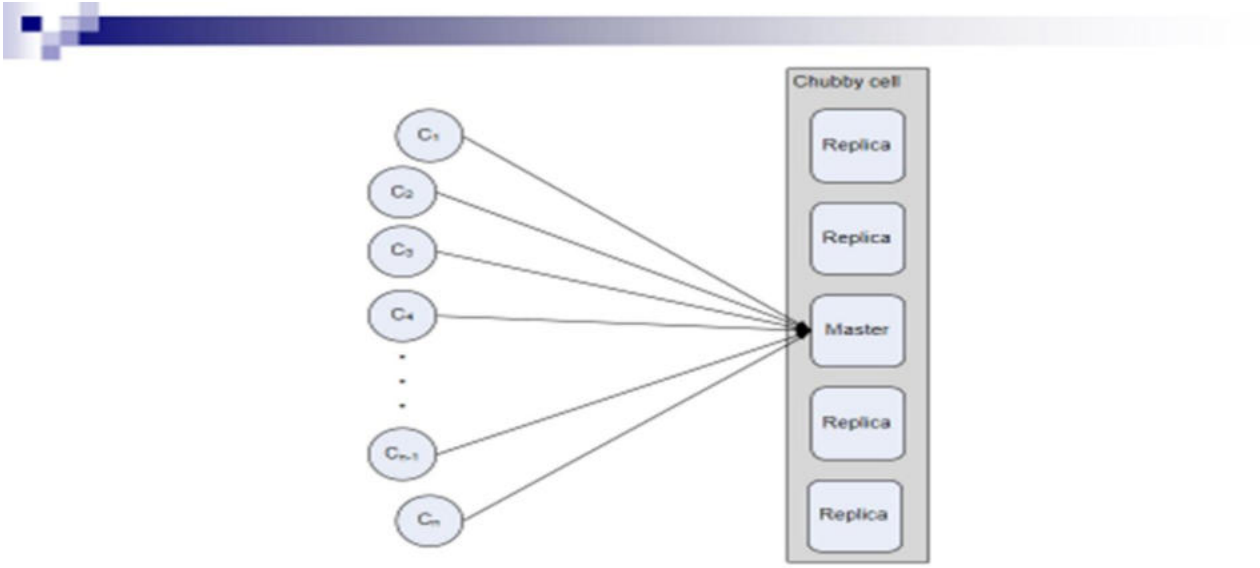
The basic organization of the system is shown in Figure 8.9.AChubby cell typically serves one data center. The cell server includes several replicas, the standard number of which is five. To reduce the probability of correlated failures, the servers hosting replicas are distributed across the campus of a data center.

The replicas use a distributed consensus protocol to elect a new master when the current one fails. The master is elected by a majority, as required by the asynchronous Paxos algorithm, accompanied by the commitment that a new master will not be elected for a period called a master lease. A session is a connection between a client and the cell server maintained over a period of time the data cached by the client, the locks acquired, and the handles of all files locked by the client are valid for only the duration of the session.

Clients use RPCs to request services from the master. When it receives a write request, the master propagates the request to all replicas and waits for a reply from a majority of replicas before responding. When it receives a read request the master responds without consulting the replicas. The client interface of the system is similar to, yet simpler than, the one supported by the Unix File System. In addition, it includes notification of events related to file or system status. A client can subscribe to events such as file content modification, change or addition of a

child node, master failure, lock acquired, conflicting lock requests, and invalid file handle.

The files and directories of the Chubby service are organized in a tree structure and use a naming scheme similar to that of Unix. Each file has a file handle similar to the file descriptor. The master of a cell periodically writes a snapshot of its dEach file or directory can act as a lock. To write to a file the client must be the only one holding the file handle, whereas multiple clients may hold the file handle to read from the file. Handles are created by a call to open() and destroyed by a call to close (). Other calls supported by the service are Get Contents And Stat(), to get the file data and meta-information, Set Contents, and Delete() and several calls allow the client to acquire and release locks. Some applications may decide to create and manipulate a sequencer with calls to Set Sequencer (), which associates a sequencer with a handle, Get Sequencer() to obtain the sequencer associated with a handle ,or check the validity of a sequencer with Check Sequencer(). At a base to a GFS file server.



- A Chubb cell consisting of 5 replicas, one of them elected as a master; n clients use RPCs to communicate with the master.

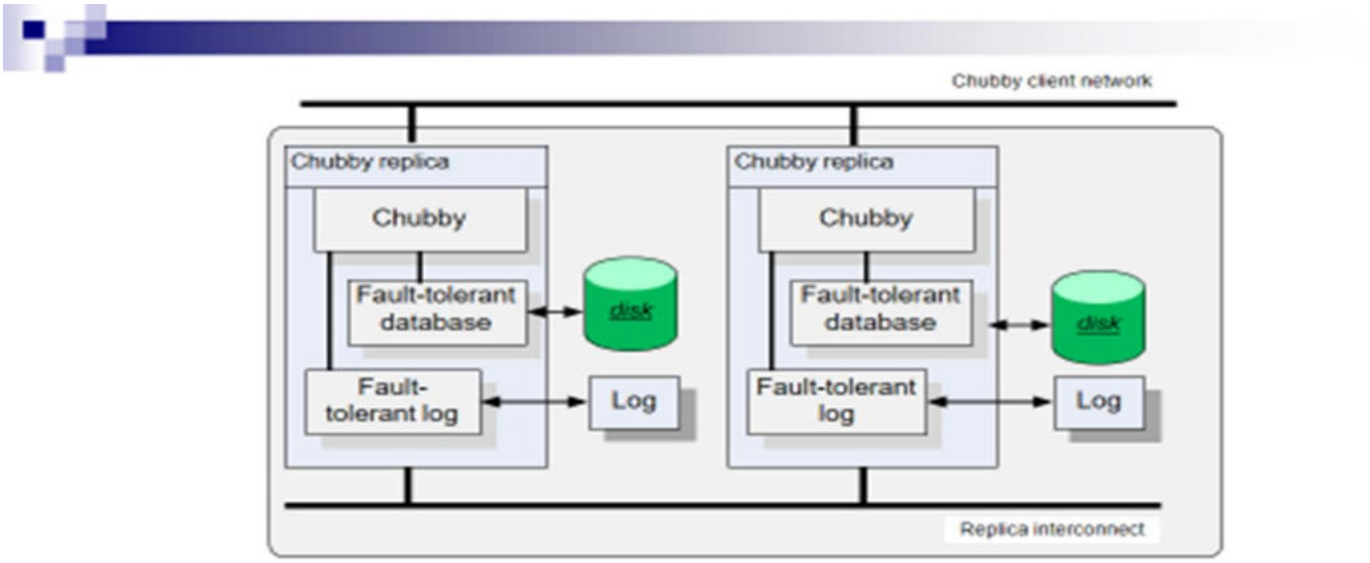
We now take a closer look at the actual implementation of the service. As pointed out earlier, Chubby locks and Chubby files are stored in a database, and this database is replicated. The architecture of these replicas shows that the stack consists of the Chubby component, which implements the Chubby protocol for communication with the clients, and the active components, which write log entries and files to the local storage of the replica see (Figure 8.10).

Recall that an atomicity log for a transaction-processing system allows a crash recovery procedure to undo all-or-nothing actions that did not complete or to finish all-or-nothing actions that committed but did not record all of their effects. Each replica maintains its own copy of the log; a new log entry is appended to the existing log and the Paxos algorithm is executed repeatedly to ensure that all replicas have the same sequence of log entries.

The next element of the stack is responsible for the maintenance of a fault-tolerant database – in other words, making sure that all local copies are consistent. The database consists of the actual data, or the local snapshot in Chubby speak, and a replay log to allow recovery in case of failure. The state of the system is also recorded in the database. The Paxos algorithm is used to reach consensus on sets of values (e.g., the sequence of entries in a replicated log). To ensure that the Paxos algorithm succeeds in spite of the occasional failure of a replica, the following three phases of the algorithm are executed repeatedly.

1. Elect a replica to be the master/coordinator. When a master fails, several replicas may decide to assume the role of a master. To ensure that the result of the election is unique, each replica generates a sequence number larger than any sequence number it has seen, in the range (1,r), where r is the number of replicas, and broadcasts it in a propose message.
2. The master broadcasts to all replicas an accept message, including the value it has selected, and waits for replies, either acknowledge or reject.

3. Consensus is reached when the majority of the replicas send an acknowledge message; then the master broadcasts the commit message.



Chubby replica architecture; the Chubby component implements the communication protocol with the clients. The system includes a component to transfer files to a fault-tolerant database and a fault-tolerant log component to write log entries. The fault-tolerant log uses the Paxos protocol to achieve consensus. Each replica has its own local file system; replicas communicate with one another using a dedicated interconnect and communicate with clients through a client network.

Transaction processing and NoSQL databases

Many cloud services are based on online transaction processing (OLTP) and operate under tight latency constraints.Moreover, these applications have to deal with extremely high data volumes and are expected to provide reliable services for very large communities of users. It did not take very long for companies heavily involved in cloud computing, such as Google and Amazon, e-commerce companies such as eBay, and social media networks such as Facebook, Twitter, or LinkedIn, to discover that traditional relational databases are not able to handle the massive amount of data and the real-time demands of online applications that are critical for their business models.

A major concern for the designers of OLTP systems is to reduce the response time. The term mem caching refers to a general-purpose distributed memory system that caches objects in main memory (RAM) the system is based on a verylarge hash table distributed across many servers. The mem cached system is based on a client-server architecture and runs under several operating systems, including Linux, Unix, Mac OS X, and Windows. The servers maintain a key-value associative array. The API allows the clients to add entries to the array and to query it. A key can be up to 250bytes long, and a value can be no larger than 1MB. The mem cached system uses an LRU cache-replacement strategy. Scalability is the other major concern for cloud OLTP applications and implicitly for data stores.

Scalability is the other major concern for cloud OLTP applications and implicitly for data stores. There is a distinction between vertical scaling, where the data and the workload are distributed to systems that share resources such as cores and processors, disks, and possibly RAM, and horizontal scaling, where the systems do not share either primary or secondary storage.

The “soft-state” approach in the design of NoSQL allows data to be inconsistent and transfers the task of implementing only the subset of the ACID properties required by a specific application to the application developer. The NoSQL systems ensure that data will be “eventually consistent” at some future point in time instead of enforcing consistency at the time when a transaction is “committed.” Data partitioning among multiple storage servers and data replication are also tenets of the NoSQL philosophy7; they increase availability, reduce response time, and enhance scalability.

The overhead of OLTP systems is due to four sources with equal contribution: logging, locking, latching, and buffer management. Logging is expensive because traditional data bases require transaction durability; thus, every write to the database can be completed only after the log has been updated. To guarantee

atomicity, transactions lock every record, and this requires access to a lock table. Many operations require multi-threading, and the access to shared data structures, such as lock tables, demands short-term latches8 for coordination. The breakdown of the instruction count for these operations in existing DBMSs is as follows: 34.6% for buffer management, 14.2% for latching, 16.3% for locking, 11.9% for logging, and 16.2% for hand-coded optimization.

Today OLTP databases could exploit the vast amounts of resources of modern computing and communication systems to store the data in main memory rather than rely on disk-resident B-trees and heap files, locking-based concurrency control, and support for multithreading optimized for the computer technologyofpastdecades[157].Logless,single-threaded,andtransaction-less data bases could replace the traditional ones for some cloud applications.

Data replication is critical not only for system reliability and availability, but also for its performance. In an attempt to avoid catastrophic failures due to power blackouts, natural disasters, or other causes (see also Section 1.6), many companies have established multiple data centers located in different geographic regions. Thus, data replication must be done over a wide area network (WAN). This could be quite challenging, especially for log data, metadata, and system configuration information, due to increased probability of communication failure and larger communication delays. Several strategies are possible, some based on master/slave configurations, and others based on homogeneous replica groups.

Master/slave replication can be asynchronous or synchronous. In the first case the master replicates write-ahead log entries to at least one slave, and each slave acknowledges appending the log record as soon as the operation is done. In the second case the master must wait for the acknowledgments from all slaves before proceeding. Homogeneous replica groups enjoy shorter latency and higher

availability than master/slave configurations. Any member of the group can initiate mutations that propagate asynchronously.

BigTable

BigTable is a distributed storage system developed by Google to store massive amounts of data and to scale up to thousands of storage servers . The system uses the Google File System discussed in Section 8.5 to store user data as well as system information. To guarantee atomic read and write operations, it uses the Chubby distributed lock service the directories and the files in the namespace of Chubby are used as locks

The system is based on a simple and flexible data model. It allows an application developer to exercise control over the data format and layout and reveals data locality information to the application clients. Any read or write row operation is atomic, even when it affects more than one column. The column keys identify column families, which are units of access control. The data in a column family is of the same type. Client applications written in C++ can add or delete values, search for a subset of data, and look up data in a row.

A row key is an arbitrary string of up to 64KB, and a row range is partitioned into tablets serving as units for load balancing. The time stamps used to index various versions of the data in a cell are 64-bit integers; their interpretation can be defined by the application, whereas the default is the time of an event in microseconds. A column key consists of a string defining the family name, a set of printable characters, and an arbitrary string as qualifier.

The organization of a BigTable (see Figure 8.11) shows a sparse, distributed, multidimensional map for an email application. The system consists of three major components :a library linked to application clients to access the system, a master server, and a large number of tablet servers. The master server controls the entire system, assigns tablets to tablet servers and balances the load among them,

manages garbage collection, and handles table and column family creation and deletion.

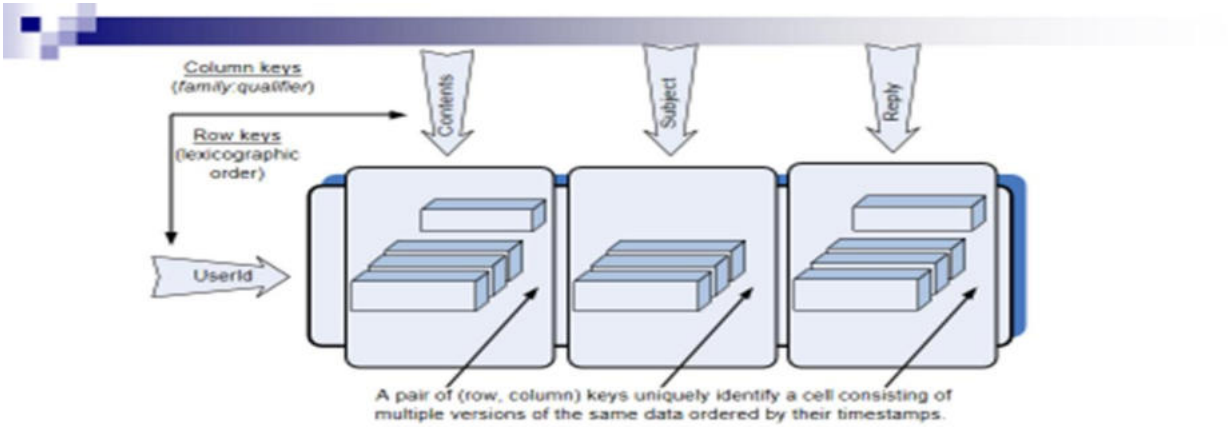
Internally, the space management is ensured by a three-level hierarchy: the root tablet, the location of which is stored in a Chubby file, points to entries in the second element, the metadata tablet, which, in turn, points to user tablets, collections of locations of users' tablets. An application client searches through this hierarchy to identify the location of its tablets and then caches the addresses for further use.

The performance of the system reported in [73] is summarized in Table 8.2. The table shows the number of random and sequential read and write and scan operations for 1,000 bytes, when the number of servers increases from 1 to 50, then to 250, and finally to 500. Locking prevents the system from achieving a linear speed-up, but the performance of the system is still remarkable due to a fair number of optimizations. For example, the number of scans on 500 tablet servers is 7 ,843/2×103instead of15 ,385/2×103.Itisreportedthat only 12 clusters use more than 500 tablet servers, whereas some 259 clusters use between 1 and 19 tablet servers.

BigTable is used by a variety of applications, including Google Earth, Google Analytics, Google Finance, and Web crawlers. For example, Google Earth uses two tables, one for preprocessing and one for serving client data. The preprocessing table stores raw images; the table is stored on disk because it contains some 70TB of data. Each row of data consists of a single image; adjacent geographic segments are stored in rows in close proximity to one another. The column family is very sparse; it contains a column for every raw image. The preprocessing stage relies heavily on MapReduce to cleanandconsolidatethedatafortheservingphase.TheservingtablestoredonGFSis"onl y"500 GB, and it is distributed across several hundred tablet servers, which

maintain in-memory column families. This organization enables the serving phase of Google Earth to provide a fast response time to tens of thousands of queries per second.

Google Analytics provides aggregate statistics such as the number of visitors to a Web page per day. To use this service, Web servers embed a JavaScript code into their Web pages to record information every time a page is visited.The data is collected in a raw-click BigTable of some 200 TB, with a row for each end-user session. A summary table of some 20TB contains predefined summaries for a Website.



The organization of an Email application as a sparse, distributed, multidimensional map. The slice of Bigtable shown consists of a row with the key *UserId* and three *family* columns; the *Contents* key identifies the cell holding the contents of Emails received, the one with key *Subject* identifies the subject of Emails, and the one with the key *Reply* identifies the cell holding the replies; the version of records in each cell are ordered according to timestamps. Row keys are ordered lexicographically; a column key is obtained by concatenating *family* and the *qualifier* fields

Big table performance – the number of operations

Number of tablet servers	Random read	Sequential read	Random write	Sequential write	Scan
1	1 212	4 425	8 850	8 547	15 385
50	593	2 463	3 745	3 623	10 526
250	479	2 625	3 425	2 451	9 524
500	241	2 469	2 000	1 905	7 843

Assignment questions for 4th one

1. Explain in detail about policies and mechanism for Resource management
2. Discuss about Network File System (NTFS) with architecture.
3. Write shot notes of the following
   a)Google File System(GFS)
   b)Big Table.

ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES
RAJAMPET
(Autonomous)

Department of Artificial Intelligence & Machine Learning

Lecture Notes

Name of the Faculty: **Dr.T.Harikrishna** **Class:** **IVYear I Semester**

Branch and Section: **AIML** **Course code:** **20A57GT**

Name of the Course: Cloud Computing

Unit-V

Unit-5

Cloud Security

The security of computing and communication systems takes on a new urgency as society becomes increasingly dependent on the information infrastructure. Nowadays, even the critical infrastructure of a nation can be attacked by exploiting flaws in computer security. Malware, such as the Stuxnet virus, targets industrial control systems controlled by software [81]. Recently, the term cyber warfare has entered the dictionary with the meaning "actions by a nation-state to penetrate another nation's computers or networks for the purposes of causing damage or disruption".

A computer cloud is a target-rich environment for malicious individuals and criminal organizations. It is thus no surprise that security is a major concern for existing users and for potential new users of cloud computing services.

Cloud computing is an entirely new approach to computing based on a new technology. It is therefore reasonable to expect that new methods to deal with some of the security threats will be developed, whereas other perceived threats will prove to be exaggerated. Indeed, "early on in the life cycle of a technology, there are many concerns about how this technology will be used ...they represent a barrier to the acceptance ...over the time, however, the concerns fade, especially if the value proposition is strong enough.

Cloud security risks

Some believe that it is very easy, possibly too easy, to start using cloud services without a proper understanding of the security risks and without the commitment to follow the ethics rules for cloud computing. A first question is: What are the security risks faced by cloud users? There is also the possibility that a cloud could be used to launch large-scale attacks against other components of the cyber infrastructure.

There are multiple ways to look at the security risks for cloud computing. A recent paper identifies three broad classes of risk: traditional security threats, threats

related to system availability, and threats related to third-party data control. Traditional threats are those experienced for some time by any system connected to the Internet, but with some cloud-specific twists. The impact of traditional threats is amplified due to the vast amount of cloud resources and the large user population that can be affected. The fuzzy bounds of responsibility between the providers of cloud services and users and the difficulties in accurately identifying the cause of a problem add to cloud users' concerns.

The traditional threats begin at the user site. The user must protect the infrastructure used to connect to the cloud and to interact with the application running on the cloud. This task is more difficult because some components of this infrastructure are outside the firewall protecting the user. The next threat is related to the authentication and authorization process. The procedures in place for one individual do not extend to an enterprise. In this case the cloud access of the members of an organization must be nuanced; individuals should be assigned distinct levels of privilege based on their roles in the organization. It is also nontrivial to merge or adapt the internal policies and security metrics of an organization with the ones of the cloud.

Moving from the user to the cloud, we see that the traditional types of attack have already affected cloud service providers. The favourite means of attack are distributed denial-of-service (DDoS) attacks, which prevent legitimate users accessing cloud services; phishing SQL injection;3 or cross-site scripting.

Cloud servers host multiple VMs, and multiple applications may run under
Each VM. Multitenency in conjunction with VMM vulnerabilities could open new attack channels for malicious users. Identifying the path followed by an attacker is much more difficult in a cloud environment. Traditional investigation methods based on digital forensics cannot be extended to a cloud, where there sources are shared among a large user population and the traces of events related to a security incident are wiped out due to the high rate of write operations on any storage media.

Availability of cloud services is another major concern. System failures, power outages, and other catastrophic events could shut down cloud services for extended periods of time.

Clouds could also be affected by phase transition phenomena and other effects specific to complex systems. Another critical aspect of availability is that users cannot be assured that an application hosted on the cloud will return correct results.

Third-party control generates a spectrum of concerns caused by the lack of transparency and limited user control. For example, a cloud provider may subcontract some resources from a third party whose level of trust is questionable. There are examples when subcontractors failed to maintain the customer data. There are also examples when the third party was not a subcontractor but a hardware supplier and the loss of data was caused by poor-quality storage devices

It is very difficult for a cloud user to prove that data has been deleted by the service provider. The lack of transparency makes auditability a very difficult proposition for cloud computing. Auditing guidelines elaborated by the National Institute of Standards and Technology (NIST), such as the Federal Information Processing Standard (FIPS) and the Federal Information Security Management Act (FISMA), are mandatory for U.S. government agencies.

The first release of the Cloud Security Alliance (CSA) report in 2010 identifies seven top threats to cloud computing. These threats are the abuse of the cloud, APIs that are not fully secure, malicious insiders, shared technology, account hijacking, data loss or leakage, and unknown risk profiles. According to this report, the IaaS delivery model can be affected by all threats. PaaS can be affected by all but the shared technology, whereas SaaS is affected by all but abuse and shared technology.

The term abuse of the cloud refers to the ability to conduct nefarious activities from the cloud – for example, using multiple AWS instances or applications supported by IaaS to launch DDoS attacks or to distribute spam and malware.

Shared technology considers threats due to multitenant access supported by virtualization. VMMs can have flaws allowing a guest operating system to affect the security of the platform shared with other virtual machines.

Insecure APIs may not protect users during a range of activities, starting with authentication and access control to monitoring and control of the application during runtime. The cloud service providers do not disclose their hiring standards and policies thus, the risks of malicious insiders cannot be ignored. The potential harm due to this particular form of attack is great.

Data loss or leakage are two risks with devastating consequences for an individual or an organization using cloud services. Maintaining copies of the data outside the cloud is often unfeasible due to the sheer volume of data. If the only copy of the data is stored on the cloud, sensitive data is permanently lost when cloud data replication fails and is followed by a storage media failure. Because some of the data often includes proprietary or sensitive data, access to such information by third parties could have severe consequences.

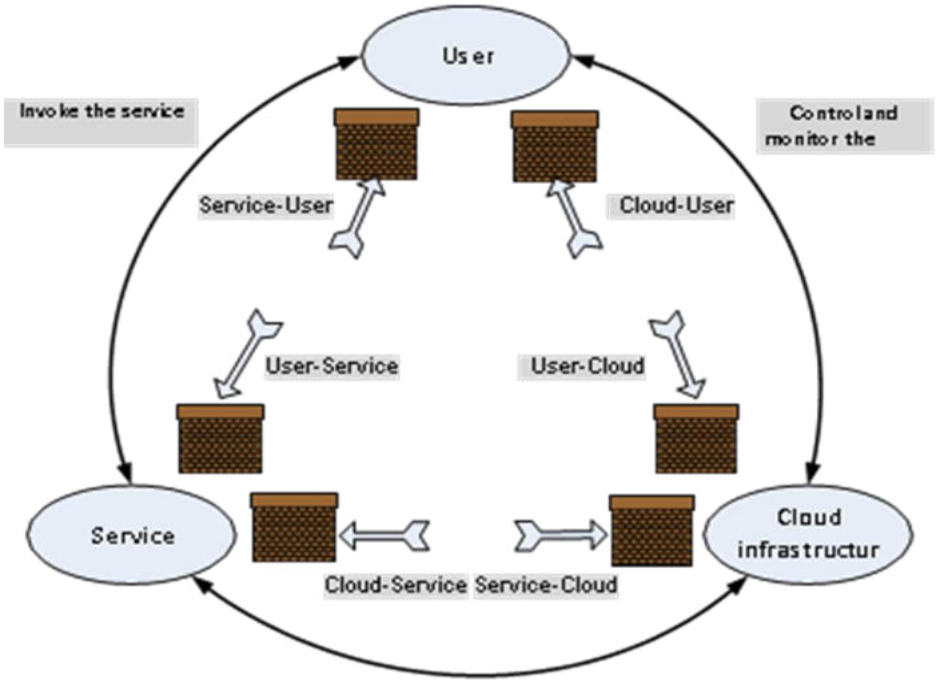
Account or service hijacking is a significant threat, and cloud users must be aware of and guard against all methods of stealing credentials. Finally, unknown risk profile refers to exposure to the ignorance or underestimation of the risks of cloud computing.

An attempt to identify and classify the attacks in a cloud computing environment is presented in. The three actors involved in the model considered are the user,the service, and the cloud infrastructure, and there are six types of attacks possible(seeFigure9.1).The user can be attacked from two directions from the service and from the cloud. SSL certificate spoofing, attacks on browser caches, or phishing attacks are examples of attacks that originate at the service.The user can also be a victim of attacks that either originate at the cloud or spoofs that originate from the cloud infrastructure.

The service can be attacked from the user. Buffer overflow, SQL injection, and privilege escalation are the common types of attacks from the service.The service

can also be subject to attack by the cloud infrastructure; this is probably the most serious line of attack. Limiting access to resources, privilege related attacks, data distortion, and injecting additional operations are only a few of the many possible lines of attack originated at the cloud.

The cloud infrastructure can be attacked by a user who targets the cloud control system. The types of attack are the same ones that a user directs toward any other cloud service. The cloud infrastructure may also be targeted by a service requesting an excessive amount of resources and causing the exhaustion of the resources.



Security: The top concern for cloud users

Users typically operate inside a secure perimeter protected by a corporate firewall.Inspite of the potential threats, users have to extend their trust to the cloud service provider if they want to benefit from the economical advantages of utility computing. This is a fairly difficult transition, yet it is a critical one for the future of cloud computing. To support this transition,some argue that cloud security is in the hands of experts,sousers are even better protected than when they are in charge of their own security.

Major user concerns are unauthorized access to confidential information and data theft. Data is more vulnerable in storage than while it is being processed. Data is kept in storage for extended periods of time, whereas it is exposed to threats during processing for relatively short periods of time. Hence close attention should be paid to the security of storage servers and to data in transit.

The next concerns regard user control over the life cycle of data. It is virtually impossible for a user to determine whether data tha tshould have been deleted is actually deleted. Even if it was deleted, there is no guarantee that the media was wiped and the next user is not able to recover confidential data. This problemisexacerbatedbecausetheCSPsrelyonseamlessbackupstopreventaccident aldataloss.Such backups are done without users' consent or knowledge. During this exercise data records can be lost, accidentally deleted, or accessible to an attacker.

Lack of standardization is next on the list of concerns. To day there are no interoperability standards, It is undeniable that auditing and compliance pose an entirely different set of challenges in cloud computing. These challenges are not yet resolved. A full audit trail on a cloud is an infeasible proposition at this time.

There is no doubt that multitenancy is the root cause of many user concerns. Nevertheless, multitenancy enables a higher server utilization thus, lower costs. Because it is one of the pillars of utility computing, users have to learn to live with multitenancy. The threats caused by multitenancy differ from one cloud delivery model to another.

Users are also greatly concerned about **the legal framework for enforcing cloud computing security**. The cloud technology has moved much faster than cloud security and privacy legislation, so users have legitimate concerns regarding the ability to defend their rights. Because the data centers of a CSP may be located in several countries, it is difficult to understand which laws apply – the laws of the country where information is stored and processed, the laws of the countries

where the information crossed from the user to the datacenter, or the laws of the country where the user is located.

Now we examine briefly what cloud users can and should do to minimize security risks regarding data handling by the CSP. First users should evaluate the security policies and the mechanisms the CSP has in place to enforce these policies. Then users should analyze the information that would be stored and processed on the cloud. Finally, the contractual obligations should be clearly spelled out. The contract between the user and the CSP should do the following.

1. State explicitly the CSP's obligations to securely handle sensitive information and its obligation to comply with privacy laws.
2. Spell out CSP liabilities for mishandling sensitive information.
3. Spell out CSP liabilities for data loss.
4. Spell out the rules governing the ownership of the data.
5. Specify the geographical regions where information and backups can be stored.

Privacy and privacy impact assessment

The term privacy refers to the right of an individual, a group of individuals, or an organization to keep information of a personal or proprietary nature from being disclosed to others. Many nations view privacy as a basic human right. The Universal Declaration of Human Rights, Article 12, states: "No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honor and reputation. Everyone has the right to the protection of the law against such interference or attacks."

Privacy concerns are different for the three cloud delivery models and also depend on the actual context. For example, consider Gmail, a widely used SaaS delivery model. Gmail privacy policy reads we collect information in two ways: information you give us ...like your name, email address, telephone number or credit card; information we get from your use of our services such as: ...device information, log.

The main aspects of privacy are: the lack of user control, potential unauthorized secondary use, data proliferation, and dynamic provisioning. The lack of user control refers to the fact that user-centric data control is incompatible with cloud usage. Once data is stored on the CSP's servers, the user loses control of the exact location, and in some instances the user could lose access to the data. For example, in case of the Gmail service the account owner has no control over where the data is stored or how long old emails are stored in some backups of the servers.

There is a need for legislation addressing the multiple aspects of privacy in the digital age. A document elaborated by the Federal Trade Commission for the U.S. Congress states "Consumer-oriented commercial Web sites that collect personal identifying information from or about consumers online would be required to comply with the four widely accepted fair information practices.

1. **Notice.**Web sites would be required to provide consumers clear and conspicuous notice of their information practices, including what information they collect, how they collect it how they use it, how they provide Choice, Access, and Security to consumers,whether they disclose the information collected to other entities, and whether other entities are collecting information through the site.
2. **Choice.**Websiteswouldberequiredtoofferconsumerschoicesastohowtheirp ersonalidentifying information is used beyond the use for which the information was provided (e.g., to consummate a transaction). Such choice would encompass both internal secondary uses (such as marketing back to consumers) and external secondary uses.
3. Access. Web sites would be required to offer consumers reasonable access to the information a Web site has collected about them, including a reason able opportunity to review information and to correct in accuracies or delete information.

4. **Security.** Web sites would be required to take reasonable steps to protect the security of the information they collect from consumers. The Commission recognizes that the implementation of these practices may vary with the nature of the information collected and the uses to which it is put, as well as with technological developments. For this reason, the Commission recommends that any legislation be phrased in general terms and be technologically neutral. Thus, the definitions of fair information practices set forth in the statute should be broad enough to provide flexibility to the implementing agency in promulgating its rules or regulations."

There is a need for tools capable of identifing privacy issues in information systems, the so-called Privacy Impact Assesment (PIA). As of mid-2012 there were no international standards for such a process, though different countries and organizations require PIA reports. An example of an analysis is to assess the legal implications of the U.K.-U.S.

API A tool that could be deployed as a Web-based service is proposed in.The inputs to the tool includes project information, an outline of project documents, privacy risks, and stakeholders. The tool will produce a PIA report consisting of a summary of findings, a risk summary, security, transparency, and cross-border data flows.

The centrepiece of the PIA tool is a knowledge base(KB) created and maintained by domain experts. The users of the SaaS service providing access to the PIA tool must fill in a questionnaire. The system usestemplatestogenerateadditionalquestionsnecessaryandtofillinthePIAreport .An expert system infers which rules are satisfied by the facts in the database and provided by the users and executes the rule with the highest priority.

Trust

Trust in the context of cloud computing is intimately related to the general problem of trust in online activities. In this section we first discuss the

traditional concept of trust and then the trust necessary to online activities. According to the Merriam-Webster dictionary, trust means “assured reliance on the character, ability, strength, or truth of someone or something.” Trust is a complex phenomenon; it enables cooperative behavior, promotes adaptive organizational forms, reduces harmful conflict, decreases transaction costs, facilitates formulation of ad hoc workgroups, and promotes effective responses to crisis.

Two conditions must exist for trust to develop. The first condition is risk, the perceived probability of loss; indeed, trust would not be necessary if there were no risk involved, if there is a certainty that an action can succeed. The second condition is interdependence, the idea that the interests of one entity cannot be achieved without reliance on other entities. A trust relationship goes through three phases: (1) a building phase, when trust is formed; (2) a stability phase, when trust exists; and (3) a dissolution phase, when trust declines.

There are different reasons for and forms of trust. Utilitarian reasons could be based on the belief that the costly penalties for breach of trust exceed any potential benefits from opportunistic behavior. This is the essence of deterrence-based trust. Another reason is the belief that the action involving the other partyisintheself-interestofthatparty.Thisistheso-calledcalculus-basedtrust.Afteralongsequence of interactions, relational trust between entities can develop based on the accumulated experience of dependability and reliance on each other.

The trust in the Internet “obscures or lacks entirely the dimensions of character and personality, nature of relationship, and institutional character” of traditional trust. The missing identity, personal characteristics, and role definitions are elements we have to deal with in the context of online trust.

Policies and reputation are two ways of determining trust. Policies reveal the conditions to obtain trust and the actions to take when some of the conditions are met. Policies require the verification of credentials.Reputation is a quality

attributed to an entity based on a relatively long history of interactions withorpossiblyobservationsoftheentity.Recommendationsarebasedontrustdec isionsmadebyothers and filtered through the perspective of the entity assessing the trust.

Operating system security

An operating system (OS) allows multiple applications to share the hardware resources of a physical system, subject to a set of policies. A critical function of an OS is to protect applications against a wide range of malicious attacks such as unauthorized access to privileged information, tempering with executable code, and spoofing. Such attacks can now target even single-user systems such as personal computers, tablets, or smartphones. Data brought into the system may contain malicious code; this could occur via a Java applet, or data imported by a browser from a malicious Web site.

The mandatory security of an OS is considered to be "any security policy where the definition of the policy logic and the assignment of security attributes is tightly controlled by a system security policy administrator". Access control authentication usage, and cryptographic usage policies are all elements of mandatory OS security. The first policy specifies how the OS controls the access to different system objects, the second defines the authentication mechanisms the OS uses to authenticate a principal, and the last specifies the cryptographic mechanisms used to protect the data.

Applications with special privileges that perform security-related functions are called trusted applications. Such applications should only be allowed the lowest level of privileges required to perform their functions. For example, type enforcement is a mandatory security mechanism that can be used to restrict a trusted application to the lowest level of privileges.

The existence of trusted paths, mechanisms supporting user interactions with trusted software, is critical to systems ecurity.If such mechanisms do not exist, malicious software can impersonate trusted software. Some systems provide

trust paths for a few functions such as log in authentication and password changing and allow servers to authenticate their clients.

A trusted-path mechanism is required to prevent malicious software invoked by an authorized application to tamper with the attributes of the object and/or with the policy rules. A trusted path is also required to prevent an impostor from impersonating the decider agent. A similar solution is proposed for cryptography usage, which should be decomposed into an analysis of the invocation mechanisms and an analysis of the cryptographic mechanism.

Specialized closed-box platforms such as the ones on some cellular phones, game consoles, and automated teller machines (ATMs) could have embedded cryptographic keys that allow themselves to reveal their true identity to remote systems and authenticate the software running on them. Such facilities are not available to open-box platforms, the traditional hardware designed for commodity operating systems.

A highly secure operating system is necessary but not sufficient unto itself; application-specific security is also necessary. Sometimes security implemented above the operating system is better. This is the case for electronic commerce that requires a digital signature on each transaction.

Virtual machine security

The hybrid and the hosted VM models in Figures 5.3(c) and (d), respectively, expose the entire system to the vulnerability of the host operating system thus, we will not analyse these models. Our discussion of virtual machine security is restricted to the traditional system VM model in Figure 5.3(b), where the VMM controls access to the hardware.

Virtual security services are typically provided by the VMM, as shown in Figure 9.2(a). Another alternative is to have a dedicated security services VM, as shown in Figure 9.2(b). A secure trusted computing base (TCB) is a necessary condition for security in a virtual machine environment; if the TCB is compromised, the security of the entire system is affected.

A guest OS runs on simulated hardware, and the VMM has access to the state of all virtual machines operating on the same hardware. The state of a guest virtual machine can be saved, restored, cloned, and encrypted by the VMM. Not only can replication ensure reliability, it can also support security, whereas cloning could be used to recognize a malicious application by testing it on a cloned system and observing whether it behaves normally. Wecan also clone a running system and examine the effect of potentially dangerous applications.

ThesecuritygroupinvolvedwiththeNISTprojecthasidentifiedthefollowingVM M-andVM-based threats:

• VMM-based threats:

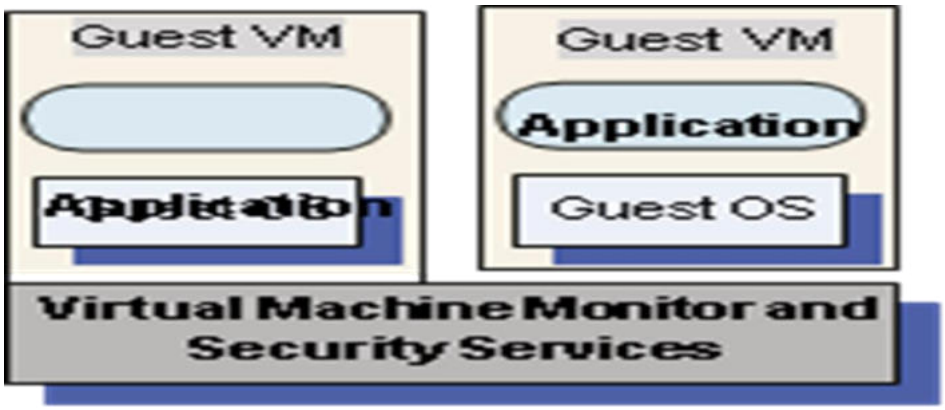
1. Starvation of resources and denial of service for some VMs.Probable causes:(a)badly configured resource limits for some VMs; (b) a rogue VM with the capability to bypass resource limits set in the VMM.
2. **VMside-channel attacks** .Malicious attacks on one or more VMs by a rogue VM under the same VMM. Probable causes: (a) lack of proper isolation of inter-VM traffic due to misconfiguration of the virtual network residing in the VMM; (b) limitation of packet inspection devices to handle high-speed traffic, e.g., video traffic; (c) presence of VM instances built from insecure VM images, e.g., a VM image having a guest OS without the latest patches.
3. Buffer overflow attacks.

VM-based threats:

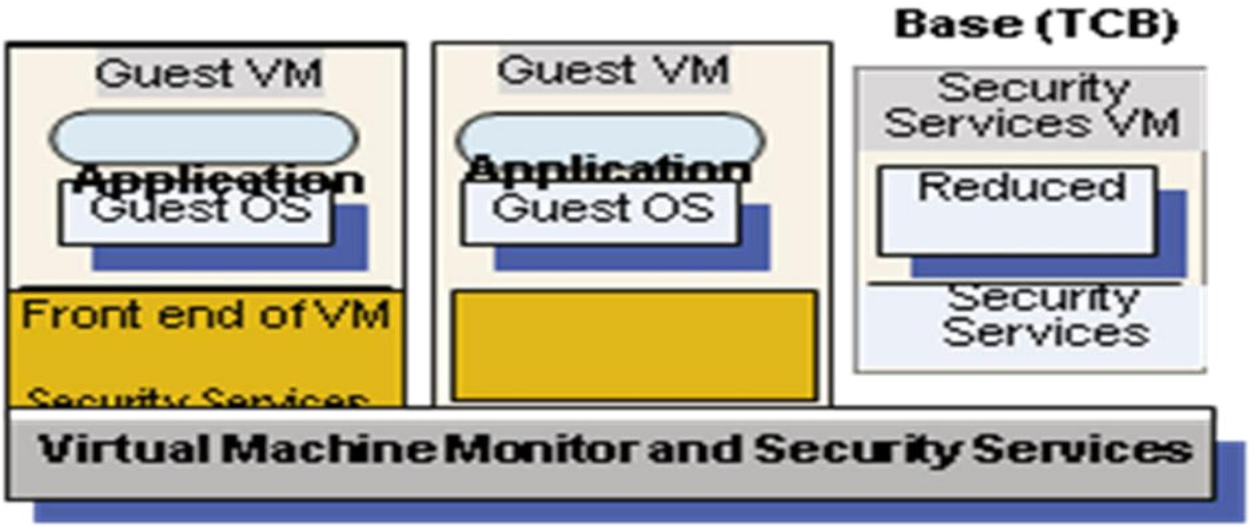
1. Deployment of rogue or insecure VM. Unauthorized users may create insecure instances from images or may perform unauthorized administrative actions on existing VMs. Probable cause: improper configuration of access controls on VM administrative tasks such as instance creation, launching, suspension, reactivation, and so on.

2. Presence of insecure and tampered VM images in the VM image repository. Probable causes: (a) lack of access control to the VM image repository; (b) lack of mechanisms to verify the integrity of the images, e.g., digitally signed image.

a) Virtual security services provided by the VMM.



b)A decidecated secured VM



Security risks posed by shared images

First, let's review the process to create an AMI. We can start from a running system, from another AMI, or from the image of a VM and copy the contents of the file system to the S3, the so-called bundling. The first of the three steps in bundling is to create an image, the second step is to compress and encrypt the

image, and the last step is to split the image into several segments and then upload the segments to the S3.

Two procedures for the creation of an image are available: ec2-bundle-image and ec2 bundle-volume. The first is used for images prepared as loop back files when the data is transferred to the image in blocks. To bundle a running system,the creator of the image can use the second procedure when bundling works at the level of the file system and files are copied recursively to the image.

Three types of security risks were analysed: (1) backdoors and leftover credentials, (2) unsolicited connections, and (3) malware. An astounding finding is that about 22% of the scanned Linux AMIs contained credentials allowing an intruder to remotely log into the system. Some 100 passwords, 995 ssh keys, and 90 cases in which both passwords and keys could be retrieved were identified.

Another threat is posed by the omission of the cloud-init script that should be invoked when the image is booted. This script, provided by Amazon, regenerates the host key an ssh server uses to identify itself; the public part of this key is used to authenticate the server. When this key is shared among several systems, these systems become vulnerable to man-in-the middle11 attacks.

Unsolicited connections pose a serious threat to a system. Outgoing connections allow an outside entity to receive privileged information, e.g., the IP address of an instance and events recorded by a syslog daemon to files in the var/log directory of a Linux system. Such information is available only to users with administrative privileges. The audit detected two Linux instances with modified syslog daemons, which forwarded to an outside agent information about events such as login and incoming requests to a Webserver.

Malware, including viruses,worms, spyware, and trojans, were identified using ClamAV, asoftware tool with a database of some 850,000 malware signatures, available from www.clamav.net. Two infected Windows AMIs were discovered, one with a Trojan-Spy(variant50112)and a second one with a Trojan-Agent(variant173287).The first Trojan carries out key logging and allows stealing

data from the files system and monitoring processes; the AMI also included a tool called Trojan.Firepass to decrypt and recover passwords stored by the Firefox browser.

Recovery of deleted files containing sensitive information poses another risk for the provider of an image. When the sectors on the disk containing sensitive information are actually over written by another file, recovery of sensitive information is much harder. To be safe, the creator of the image effort should use utilities such as shred, scrub, zero free, or wipe to make recovery of sensitive information next to impossible.

Security risks posed by a management OS

A hypervisor supports stronger isolation between the VMs running under it than the isolation between processes supported by a traditional operating system. Yet the hypervisor must rely on management OS to create VMs and to transfer data in and out from a guest VM to storage devices and network interfaces.

A small VMM can be carefully analysed thus, one could conclude that the security risks in a virtual environment are diminished. We have to be cautious with such sweeping statements. Indeed, the trusted computer base (TCB)15 of a cloud computing environment includes not only the hypervisor but also themanagementOS.ThemanagementOSsupportsadministrativetools,livemigration,devicedrivers, and device emulators.

Dom0 manages the building of all user domains (DomU), a process consisting of several steps:

1. Allocate memory in the Dom0 address space and load the kernel of the guest operating system from secondary storage.
2. Allocate memory for the new VM and use foreign mapping to load the kernel to the new VM.
3. Set up the initial page tables for the new VM.
4. Release the foreign mapping on the new VM memory, set up the virtual CPU registers, and launch the new VM.

A malicious Dom0 can play several nasty tricks at the time when it creates a DomU .

Refuse to carry out the steps necessary to start the new VM, an action that can be considered a denial-of-service attack.

• Modify the kernel of the guest operating system in ways that will allow a third party to monitor and control the execution of applications running under the new VM.

• Undermine the integrity of the new VM by setting the wrong page tables and/or setting up incorrect virtual CPU registers.

• Refuse to release the foreign mapping and access the memory while the new VM is running.

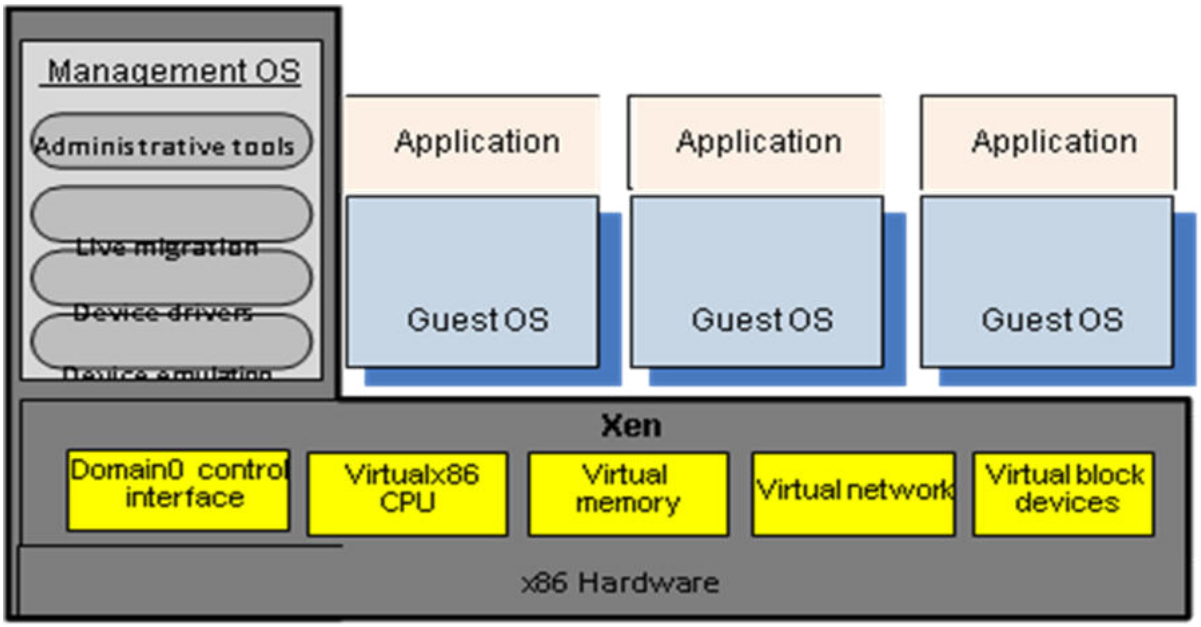
Dom0 should be prohibited from using foreign mapping for sharing memory with a DomU unless a DomU initiates the procedure in response to a hypercall from Dom0. When this happens, Dom0 should be provided with an encrypted copy of the memory pages and of the virtual CPU registers. The entire process should be closely monitored by the hypervisor, which, after the access, should check the integrity of the affected DomU.

New hypercalls are necessary to protect:

• The privacy and integrity of the virtual CPU of a VM. When Dom0 wants to save the state of the VM, the hypercall should be intercepted and the contents of the virtual CPU registers should be encrypted. When a DomU is restored, the virtual CPU context should be decrypted and then an integrity check should be carried out. •

The privacy and integrity of the VM virtual memory. The page table update hyper call should be intercepted and the page should be encrypted so that Dom0 handles only encrypted pages of the VM. To guarantee integrity, the hypervisor should calculate a hash of all the memory pages before they are saved by Dom0. Because a restored DomU may be allocated a different memory region, an address translation is necessary.

• The freshness of the virtual CPU and the memory of the VM. The solution is to add to the hash a version number.

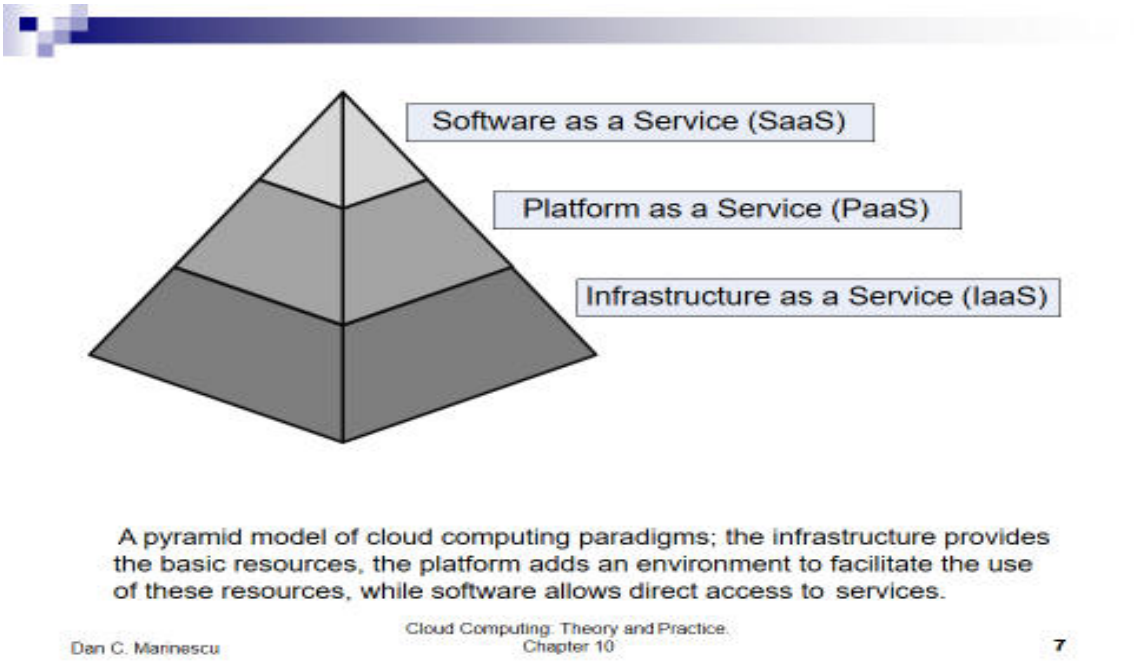


Cloud Application Development

It is fair to assume that the population of application developers and cloud users is and will continue to be very diverse. Some cloud users have developed and run parallel applications on clusters or other types of systems for many years and expect an easy transition to the cloud. Others are less experienced but willing to learn and expect a smooth learning curve. Many view cloud computing as an opportunity to develop new businesses with minimum investment in computing equipment and human resources.

The answers to these questions are different for the three cloud delivery models, SaaS, PaaS, and IaaS the level of difficulty increases as we move toward the base of the cloud service pyramid, as shown in Figure 11.1. Recall that SaaS applications are designed for end users and are accessed over the Web; in this case, users must be familiar with the API of a particular application. PaaS provides a set of tools and services designed to facilitate application coding and deploying; IaaS provides the hardware and the software for servers, storage, and networks, including operating systems and storage management software. The

IaaS model poses the most challenges; thus, we restrict our discussion to the IaaS cloud computing model and concentrate on the most popular services offered at this time, the Amazon Web Services (AWS).



Amazon Web Services:

EC2 instances Figure 11.2 displays the Amazon Management Console (AMC) window listing the Amazon Web Services offered at the time of this writing. The services are grouped into several categories: computing and networking, storage and content delivery, deployment and management, databases, and application services.

Recall that an AWS EC2 instance is a virtual server started in a region and the availability zone is selected by the user. Instances are grouped into a few classes, and each class has available to it a specific amount of resources, such as: CPU cycles, main memory, secondary storage, and communication and I/O bandwidth. Several operating systems are supported by AWS, including Amazon Linux, Red Hat Enterprise Linux, 6.3, SUSE Linux Enterprise Server 11, Ubuntu Server 12.04.1, and several versions of Microsoft Windows.

The next step is to create an (AMI) 1 on one of the platforms supported by AWS and start an instance using the Run Instance API. If the application needs more

than 20 instances, a special form must be filled out.The local instance store persists only for the duration of an instance the data will persist if an instance is started using the Amazon Elastic Block Storage(EBS) and then the instance can be restarted at a later time. Once an instance is created, the user can perform several actions – for example, connect to the instance, launch more instances identical to the current one, or create an EBS AMI. The user can also terminate, reboot, or stop the instance (see Figure 11.4). The Network & Security panel allows the creation of Security Groups, Elastic IP addresses, Placement Groups, Load Balancers, and Key Pairs whereas the EBS panel allows the specification of volumes and the creation of snapshots.

Connecting clients to cloud instances through firewalls

A firewall is a software system based on a set of rules for filtering network traffic. Its function is to protect a computer in a local area network from unauthorized access. The first generation of firewalls, deployed in the late 1980s, carried out packet filtering; they discarded individual packets that did not matchasetofacceptancerules.Suchfirewallsoperatedbelowthetransportlayeranddiscardedpackets based on the information in the headers of physical, data link, and transport layer protocols.

The second generation of firewalls operate at the transport layer and maintain the state of all connections passing through them. Unfortunately, this traffic-filtering solution opened the possibility of denial-of-service (DoS) attacks. A DoS attack targets a widely used network service and forces the operating system of the host to fill the connection tables with illegitimate entries. DoS attacks prevent legitimate access to the service.

The third generation of firewalls “understand” widely used application layer protocols such as FTP, HTTP, TELNET, SSH, and DNS. These firewalls examine the header of application layer protocols and support intrusion detection systems (IDSs). Firewalls screen incoming traffic and sometimes filter outgoing traffic as well. A first filter encountered by the incoming traffic in a typical network is a

firewall provided by the operating system of the router; the second filter is a firewall provided by the operating system running on the local computer (see Figure 11.5).

Typically, the local area network (LAN) of an organization is connected to the Internet via a router. A router firewall often hides the true address of hosts in the local network using the Network Address Translation (NAT) mechanism. The hosts behind a firewall are assigned addresses in a “private address range,”and the router uses the NAT tables to filter the incoming traffic and translate external IP addresses to private ones.

If one tests a client-server application with the client and the server in the same LAN, the packets do not cross a router. Once a client from a different LAN attempts to use the service, the packets may be discarded by the router’s firewall. The application may no longer work if the router is not properly configured.

A rule specifies a filtering option at (i) the network layer, when filtering is based on the destination/source IP address; (ii) the transport layer, when filtering is based on destination/source port number; or (iii) the MAC layer, when filtering is based on the destination/source MAC address.

In Linux or Unix systems the firewall can be configured only as a root using the sudo command. The firewall is controlled by a kernel data structure, the ip tables.The ip tables command is used to set up, maintain, and inspect the tables of the IPv4 packet filter rules in the Linux kernel. Several tables may be defined; each table contains a number of built-in chains and may also contain user-defined chains.A chain is a list of rules that can match a set of packets: The INPUT rule controls all incoming connections the FORWARD rule controls all packets passing through this host; and the OUTPUT rule controls all outgoing connections from the host. A rule specifies what to do with a packet that matches: Accept, let the packet pass; Drop, discharge the packet; Queue, pass the packet to the user space; or Return, stop traversing this chain and resume processing at the head of the next chain. For complete information on the iptables, see

http://linux.die.net/man/8/iptables. To get the status of the firewall, specify the L (List) action of the iptables command:

sudo iptables –L

To change the default behavior for the entire chain, specify the action P (Policy), the chain name, and the target name; e.g., to allow all outgoing traffic to pass unfiltered, use sudo iptables -P OUTPUT ACCEPT s To add a new security rule, specify: the action, A (add), the chain, the transport protocol, TCP or UDP, and the target ports, as in:

sudo iptables -A INPUT -p -tcp -dport ssh -j ACCEPT

sudo iptables -A OUTPUT -p -udp -dport 4321 -j ACCEPT

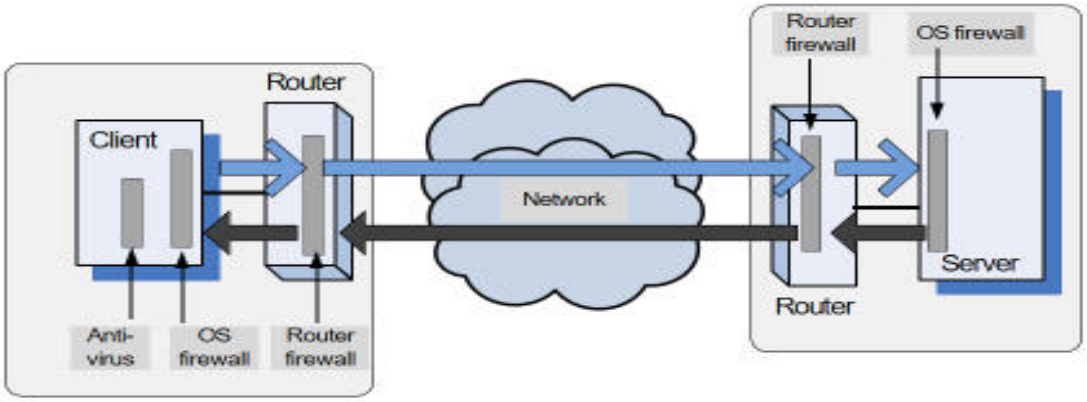
sudo iptables -A FORWARD -p -tcp -dport 80 -j DROP

To delete a specific security rule from a chain, set the action D (Delete) and specify the chain name and the rule number for that chain. The top rule in a chain has number 1:

sudo iptables -D INPUT 1

sudo iptables -D OUTPUT 1

sudo iptables -D FORWARD 1



Firewalls screen incoming and sometimes outgoing traffic. The first obstacle encountered by the inbound or outbound traffic is a router firewall, the next one is the firewall provided by the host operating system; sometimes, the antivirus software provides a third line of defense.

Important question for 5th Unit

1. Discuss about threats in cloud service
2. Explain about services provided by AWS EC2 instances
3. Discuss about connecting clients to cloud instances through firrewalls.